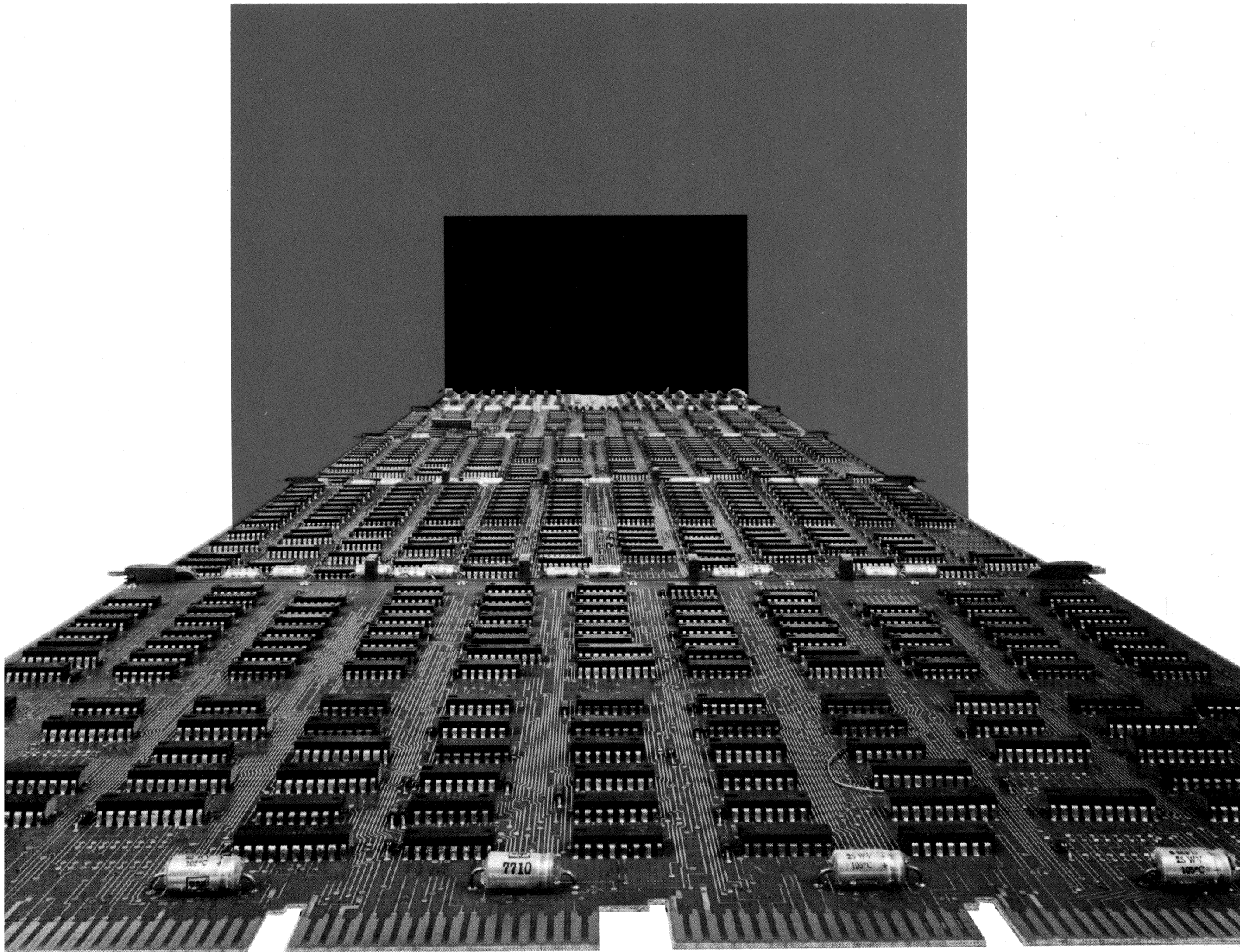


VAX-11/780

Processor Internals



digital

BLANK

VAX-11/780

Processor Internals

Student Workbook

COURSE NUMBER
EY-C5114-LE-001
(J6024-A)

Copyright © 1980, Digital Equipment Corporation.
All Rights Reserved.

The reproduction of this material, in part or whole, is strictly prohibited. For copy information, contact the Educational Services Department, Digital Equipment Corporation, Bedford, Massachusetts 01730.

Printed in U.S.A.

The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation. Digital Equipment Corporation assumes no responsibility for any errors that may appear in this document.

The software described in this document is furnished under a license and may not be used or copied except in accordance with the terms of such license.

Digital Equipment Corporation assumes no responsibility for the use or reliability of its software on equipment that is not supplied by Digital.

The following are trademarks of Digital Equipment Corporation, Maynard, Massachusetts:

DIGITAL	DECsystem-10	MASSBUS
DEC	DECSYSTEM-20	OMNIBUS
PDP	DIBOL	OS/8
DECUS	EDUSYSTEM	RSTS
UNIBUS	VAX	RSX
	VMS	IAS

CONTENTS

STUDENT GUIDE

COURSE DESCRIPTION	SG-1
COURSE GOALS	SG-1
VAX 11/780 SYSTEM BLOCK DIAGRAM.	SG-2
COURSE MAP	SG-3
COURSE OUTLINE	SG-5
RESOURCES.	SG-7
TESTING CRITERIA	SG-8

PROCESS ENVIRONMENT

INTRODUCTION.	1-1
OBJECTIVES.	1-1
SAMPLE TEST ITEM.	1-2
RESOURCES	1-2
LECTURE OUTLINE	1-3
CLASSROOM VISUALS	1-5
APPENDIX A - CONTENTS of the SOFTWARE PCB	1-13
APPENDIX B - FIXED PORTION of PROCESS HEADER.	1-17
APPENDIX C - PROCESS ENVIRONMENT.	1-23
Introduction.	1-23
Hardware Context.	1-24
Software Context.	1-24
Virtual Address Space	1-25
Program Region.	1-25
Control Region.	1-26
System Region	1-26
MODULE TEST	1-27

CONTENTS (Cont)

ADDRESS TRANSLATION

INTRODUCTION.	2-1
OBJECTIVES.	2-1
SAMPLE TEST ITEM.	2-2
RESOURCES	2-2
LECTURE OUTLINE	2-3
CONCEPTS.	2-5
Protection.	2-5
Mapping	2-6
Physical Address Space.	2-12
PAGE TABLES and MAPPING REGISTERS	2-13
Page Tables	2-13
Page Table Entries.	2-15
Base and Length Registers	2-22
MEMORY MANAGEMENT EXCEPTIONS.	2-26
Access Violation.	2-27
Translation-Not-Valid Fault	2-28
ADDRESS TRANSLATION	2-29
System Virtual Address Translation.	2-29
Process Virtual Address Translation	2-31
Translation Buffer.	2-37
Address Translation Faults.	2-39
APPENDIX A - TRANSLATION BUFFER	2-43
APPENDIX B - WORKSHEETS	2-47
MODULE TEST	2-53

CONSOLE INTERFACE BOARD

INTRODUCTION.	3-1
OBJECTIVES.	3-1
SAMPLE TEST ITEMS	3-2
RESOURCES	3-2

CONTENTS (Cont)

LECTURE OUTLINE	3-3
CLASSROOM VISUALS	3-5
LAB EXERCISES	3-9
MODULE TEST	3-13

MICROCONTROL SUBSYSTEM

INTRODUCTION.	4-1
OBJECTIVES.	4-2
SAMPLE TEST ITEMS	4-2
RESOURCES	4-2
LECTURE OUTLINE	4-3
CLASSROOM VISUALS	4-5
EXERCISES	4-11
MODULE TEST	4-31

11/780 INTERNAL OPERATIONS

INTRODUCTION.	5-1
OBJECTIVES.	5-1
SAMPLE TEST ITEMS	5-2
RESOURCES	5-2
LECTURE OUTLINE	5-3
SUPPLEMENT A - VAX-11/780 MICROSEQUENCES.	5-43
SUPPLEMENT B - V-BUS LISTINGS	5-49
SUPPLEMENT C - MICROSTATES for PROGRAM UNDER STUDY. . .	5-55
SUPPLEMENT D - DISCUSSION of MICROSTATE 62.	5-57
SUPPLEMENT E - COMMAND FILE for PROGRAM UNDER STUDY . .	5-63

CONTENTS (Cont)

SUPPLEMENT F - VAX-11/780 JARGON.	5-67
SUPPLEMENT G - INSTRUCTION REGISTER DECODE WORKSHEETS .	5-73
SUPPLEMENT H - INSTRUCTION BUFFER TRUTH TABLES.	5-77
Tables for IDPH SEL B2-6 S1/S0H	5-78
Tables for Valid Bit Shifters	5-79
CLASSROOM VISUALS	5-84
LAB EXERCISES	5-95
MODULE TEST	5-109

WCS DEBUGGER

INTRODUCTION.	6-1
OBJECTIVES.	6-1
SAMPLE TEST ITEM.	6-1
RESOURCES	6-2
LECTURE OUTLINE	6-2
XFC OP CODE PROGRAM	6-3
LAB EXERCISES	6-5
MODULE TEST	6-7

MICRODIAGNOSTICS

INTRODUCTION.	7-1
OBJECTIVES.	7-1
SAMPLE TEST ITEM.	7-1
RESOURCES	7-2
LECTURE OUTLINE	7-2
MODULE TEST	7-3

CONTENTS (Cont)

FLOATING-POINT ACCELERATOR

INTRODUCTION.	8-1
OBJECTIVES.	8-1
SAMPLE TEST ITEM.	8-2
RESOURCES	8-2
LECTURE OUTLINE	8-3
FLOATING-POINT ACCELERATOR.	8-5
FPA Installation.	8-6
FPA/CPU Interface	8-7
Microstates for FPA	8-8
FPA Floating-Point Multiply Instruction	8-9
Comments for the ROM States in WCS.	8-11
FPA ROM Word.	8-12
Example of FPA Multiply Algorithm	8-14
EXERCISES	8-15
MODULE TEST	8-17

MS-780 MEMORY SYSTEM

INTRODUCTION.	9-1
OBJECTIVES.	9-1
SAMPLE TEST ITEM.	9-1
RESOURCES	9-2
LECTURE OUTLINE	9-3
EXERCISES	9-5
CLASSROOM VISUALS	9-19
MODULE TEST	9-25

CONTENTS (Cont)

RH-780 MASSBUS ADAPTER

INTRODUCTION	10-1
OBJECTIVES	10-1
SAMPLE TEST ITEM	10-1
RESOURCES.	10-1
LECTURE OUTLINE.	10-2
CLASSROOM VISUALS.	10-3
MODULE TEST	10-19

UNIBUS ADAPTER

INTRODUCTION	11-1
OBJECTIVES	11-1
SAMPLE TEST ITEM	11-2
RESOURCES.	11-2
LECTURE OUTLINE.	11-3
EXERCISES.	11-5
LAB EXERCISES.	11-7
MODULE TEST	11-55

STUDENT GUIDE

COURSE DESCRIPTION

During a five-week training period the student will acquire an in-depth working knowledge of the internal hardware and firmware needed to diagnose failures that are detected but not isolated by the diagnostics.

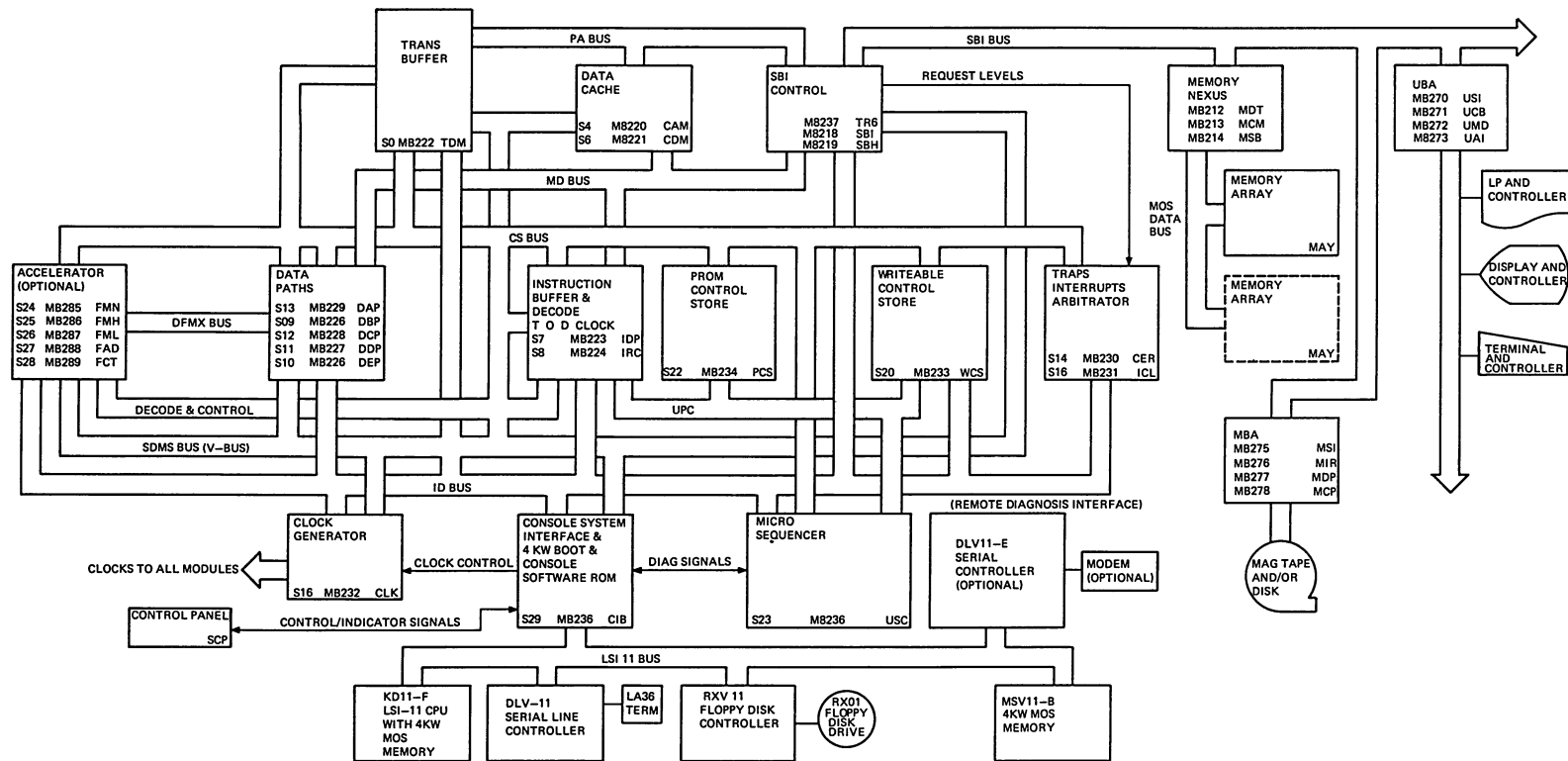
The hardware covered in this course includes the KA780 processor, MS780 memory, RH780 MASSBUS adapter, and the DW780 UNIBUS adapter.

The theory of operation will be covered in detail during classroom lecture. Key topics will receive further emphasis in the laboratory.

The lecture/lab ratio is 80/20.

COURSE GOALS

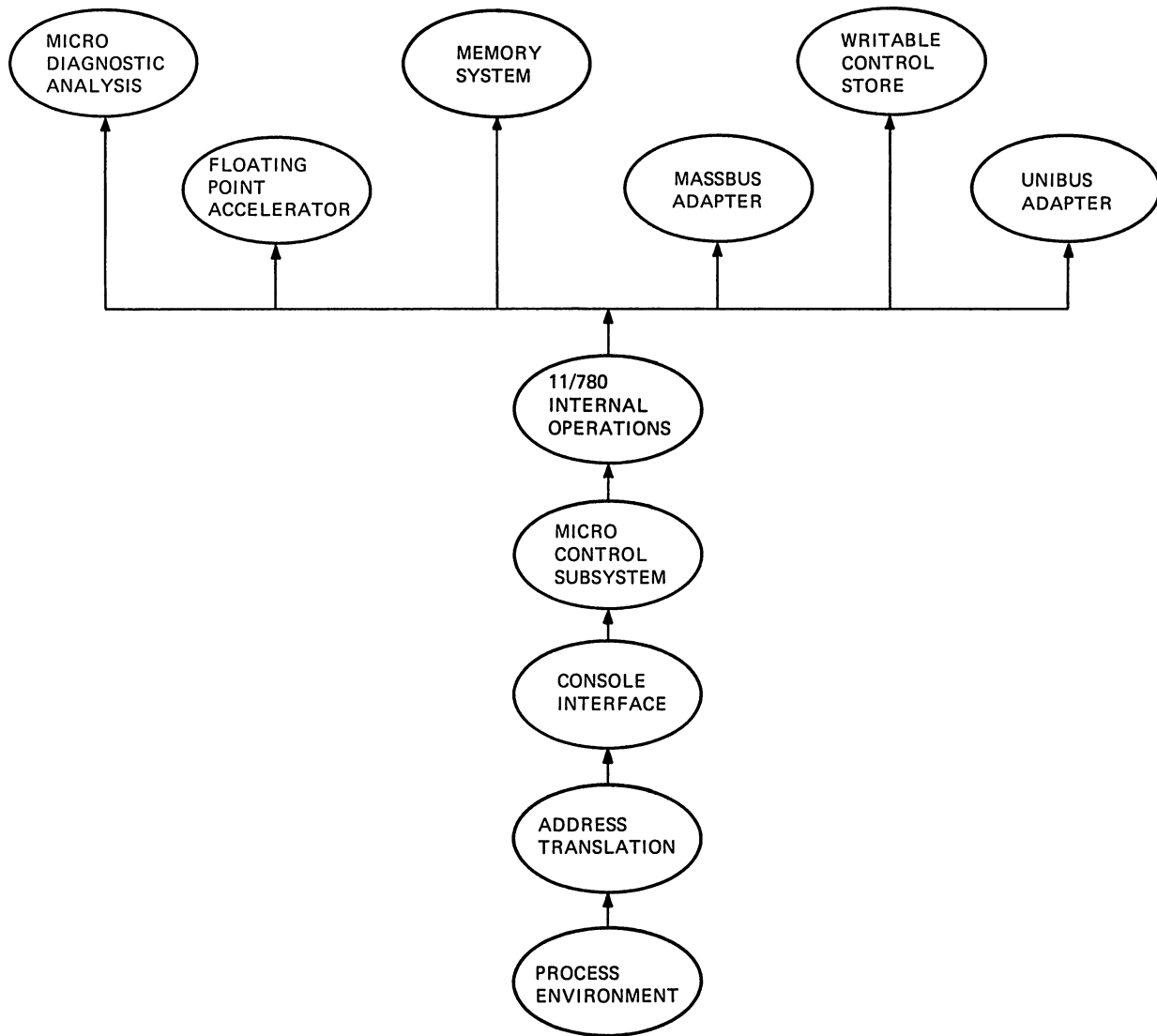
1. Using WCS debugger, develop, load, and execute a small (up to five microinstruction) routine to perform a specified function.
2. Demonstrate console and ID bus functions at the LSI ODT level.
3. Given a microdiagnostic error printout of an error that is detected but not isolated, use the microdiagnostic listings to identify the failing module within the CPU.
4. Given a macrodiagnostic error printout of an error that is detected but not isolated, use the macrolistings to identify the failing module within the CPU cluster.



TK-4358

VAX-11/780 System Block Diagram

COURSE MAP



TK-4350

BLANK

COURSE OUTLINE

- I. Process Environment
 - A. Definition of Process
 - B. Hardware Context (Hardware PCB)
 - C. Software Context (Software PCB)
 - D. Virtual Address Space
 - E. Loading of Process
- II. Address Translation
 - A. Concept
 - B. Page Tables and Mapping Registers
 - C. Memory Management Exceptions
 - D. Address Translation Process
- III. Console Interface Board (CIB)
 - A. Block Diagram Functional Overview
 - B. Functions Implemented by Hardware
 - C. Functions Implemented by Microcode
- IV. Microcontrol Subsystem
 - A. Block Diagram Functional Overview
 - B. Interpretation of Microword Fields
 - C. Microaddress Generation
- V. 11/780 Internal Operations

The 11/780 Internal Operations module is a series of in-depth discussions which will aid the student's understanding of the hardware performance for the process that was developed by the instructor in the first two days of the course. The discussions will be supplemented by in-class work assignments and laboratory work and will cover all the remaining hardware modules.
- VI. Writable Control Store (WCS)
 - A. Block Diagram Functional Overview
 - B. Write WCS Using Floppy File
 - C. Write WCS Using VAX Macrocode

VII. Microdiagnostics

- A. Interpreting Microdiagnostic Listing
- B. Microdiagnostic Fault Analysis

VIII. Floating-Point Accelerator (FPA)

- A. Floating-Point Format
- B. Floating-Point Math Concepts
- C. Block Diagram Functional Overview
- D. FPA Control Interface Signals with CPU
- E. FPA Add/Logic
- F. FPA Multiply/Logic

IX. Memory Subsystems

- A. Block Diagram Functional Overview
- B. Memory Controller/Registers
- C. Memory Arrays
- D. Memory Flows/Logic
- E. Error Detection and Correction Logic

X. MASSBUS Adapter (MBA)

- A. Block Diagram Functional Overview
- B. Programming Description
- C. MASSBUS Flows and Related Logic

XI. UNIBUS Adapter (UBA)

- A. Block Diagram Functional Overview
- B. Programming Description
- C. UNIBUS Flows and Related Logic

RESOURCES

1. VAX-11/780 Architecture Handbook
2. VAX-11/780 Hardware Handbook
3. VAX-11/780 Software Handbook
4. VAX-11/780 System Maintenance Guide
5. Computer Interfacing Accessories and Logic Handbook
6. Microcomputer Processor's Handbook
7. PDP-11 Peripheral Handbook
8. Memories and Peripheral Handbook
9. VAX-11/780 Diagnostic System User's Guide
10. VAX-11/780 Hardware User's Guide
11. VAX-11/780 Technical Summary
12. Print Sets
 - a. RH 780
 - b. DW 780
 - c. KA 780
 - d. KC 780
 - e. Unit Assy 11/780
 - f. Power System
13. VAX/VMS Command Language User's Guide
14. Microfiche Library
15. LSI-11 Maintenance Card
16. Introduction to VAX 11: Concepts audio/visual course
17. KA780 Central Processor
18. KC780 Console Interface Technical Description
19. TB/CACHE/SBI Control Technical Description
20. RH780 MASSBUS Adapter Technical Description
21. DW780 UNIBUS Adapter Technical Description

- 22. MS780 Memory System Technical Description
- 23. FP780 Floating-Point Accelerator Technical Description
- 24. Diagnostic System Technical Description
- 25. Power System Technical Description

TESTING CRITERIA

The student must take a test at the end of each module. The passing grade will be a minimum of 70%.

If a student does not make a passing grade on any one module, the instructor will discuss the situation with his supervisor to decide a course of action.

PROCESS ENVIRONMENT

1

INTRODUCTION

In this module the student will be introduced to process environment software concepts and terminology. The terminology relating to process environment will aid the student to interpret program control store (PCS) listing comments in future lessons.

OBJECTIVES

1. Identify the component parts of a process and where they reside.
2. Identify the component parts of the hardware context.

SAMPLE TEST ITEMS

1. The hardware context of a process is a combination of the contents of the general purpose registers and the process registers that are process specific.

Indicate by placing an X to the left of the registers that are part of the hardware context.

Note: Not all hardware context registers are listed.

___	System control block base register
___	Frame pointer register
___	Interrupt priority level register
___	System length register
___	Argument pointer register
___	Processor status longword register
___	Software interrupt request register
___	Asynchronous system trap level register
___	Interrupt stack pointer register
___	P0 base register
___	Process control block base register
___	SBI fault/status register
___	P1 length register
___	System length register
___	Kernel stack pointer register

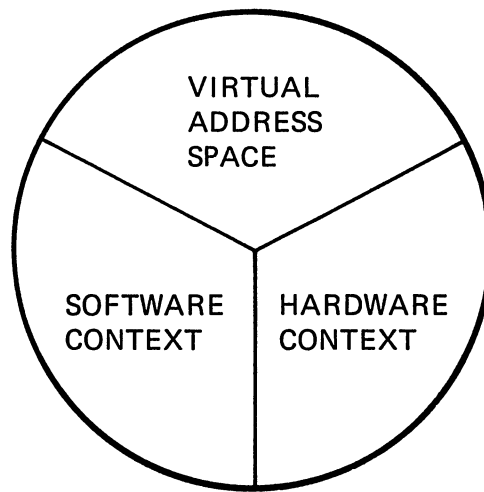
RESOURCES

1. VAX-11/780 Architecture Handbook, Chapters 3 and 13
2. VAX-11/780 Hardware Handbook, Chapter 4
3. VAX-11/780 Software Handbook, Sections 2.3 and 2.4

LECTURE OUTLINE

- I. Definition of a Process
 - A. A process is the basic schedulable entity in the VAX/VMS system.
 - B. Components of a process
- II. Hardware Context
- III. Software Context (Software PCB)
- IV. Virtual Address Space
 - A. Process header - a data structure in system space that is both paged and swapped
 - B. Program region (P0 space) - contains the program code
 - C. Control region (P1 space)
 - D. System region - contains the operating system components
- V. Loading the Process
 - A. Operating system must remove previous process
 - B. Operating system must load current process
 - C. Sequence of events (big picture)

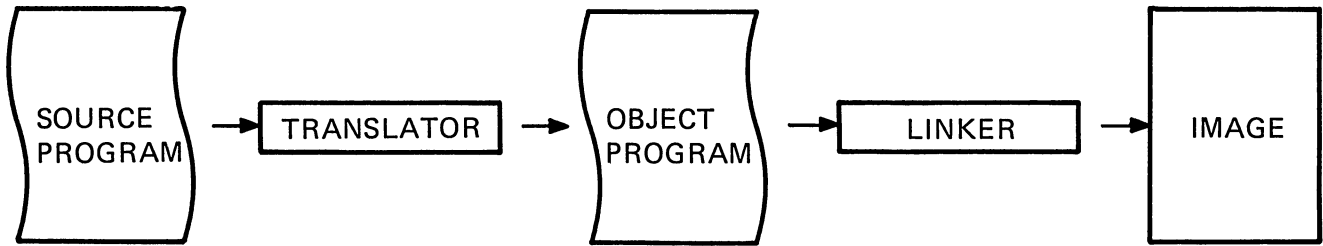
BLANK



A1

TK-3513

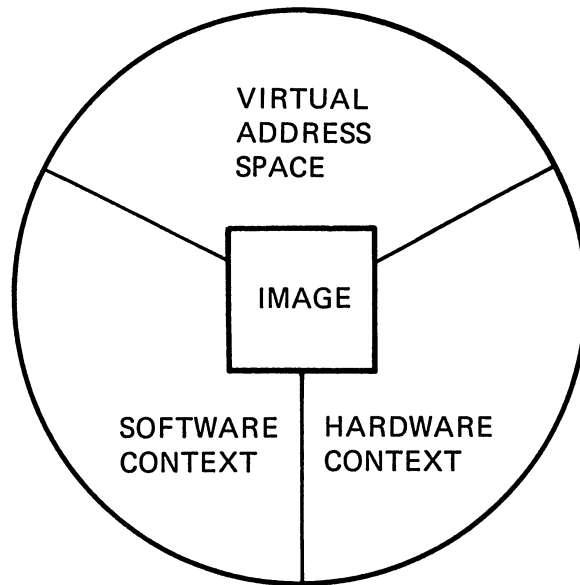
Figure 1-1. Process



A2

TK-3514

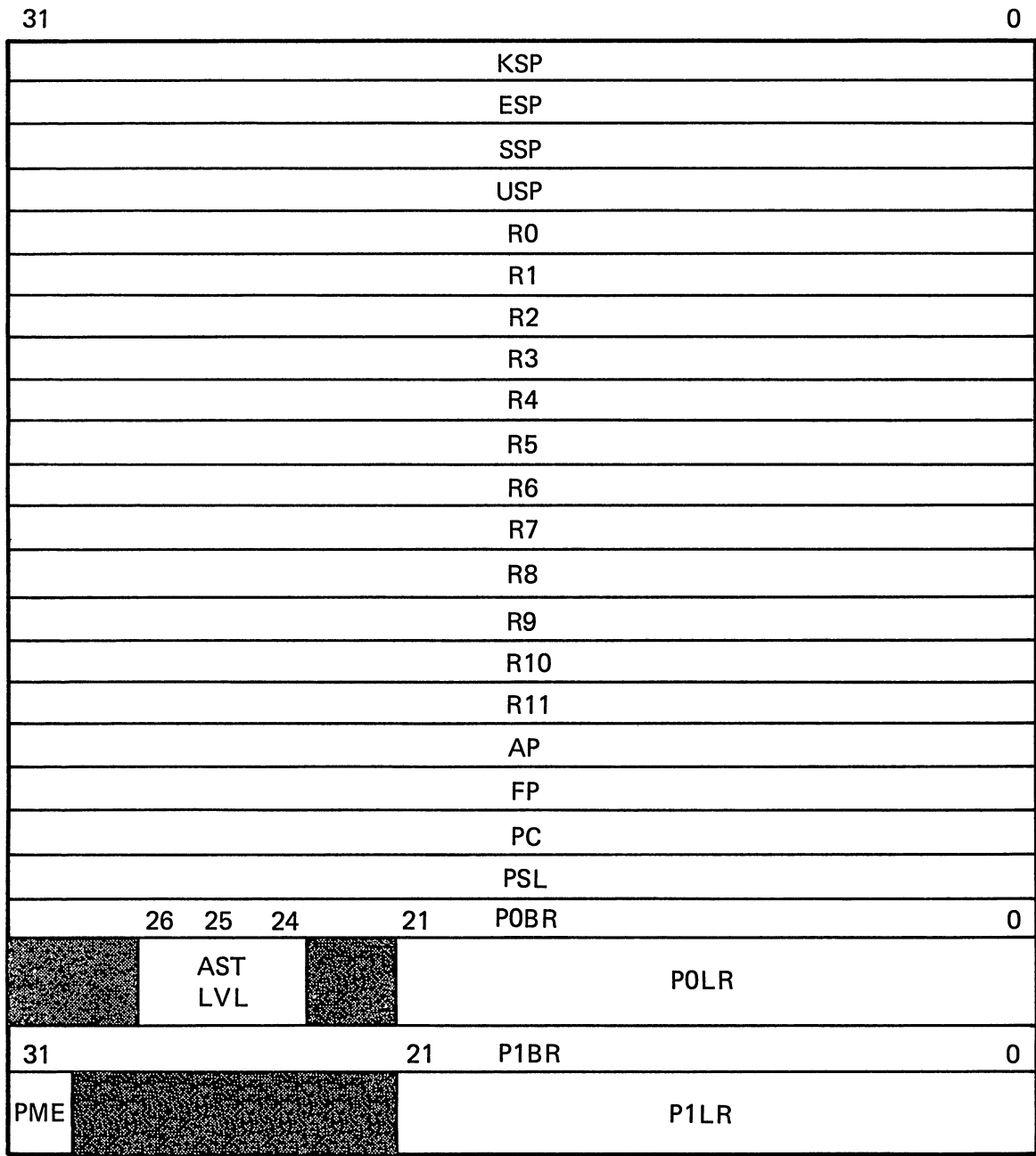
Figure 1-2. Creating an Image



A3

TK-3515

Figure 1-3. Image Within a Process



96 BYTES

Figure 1-4. Hardware Process Control Block (PCB)

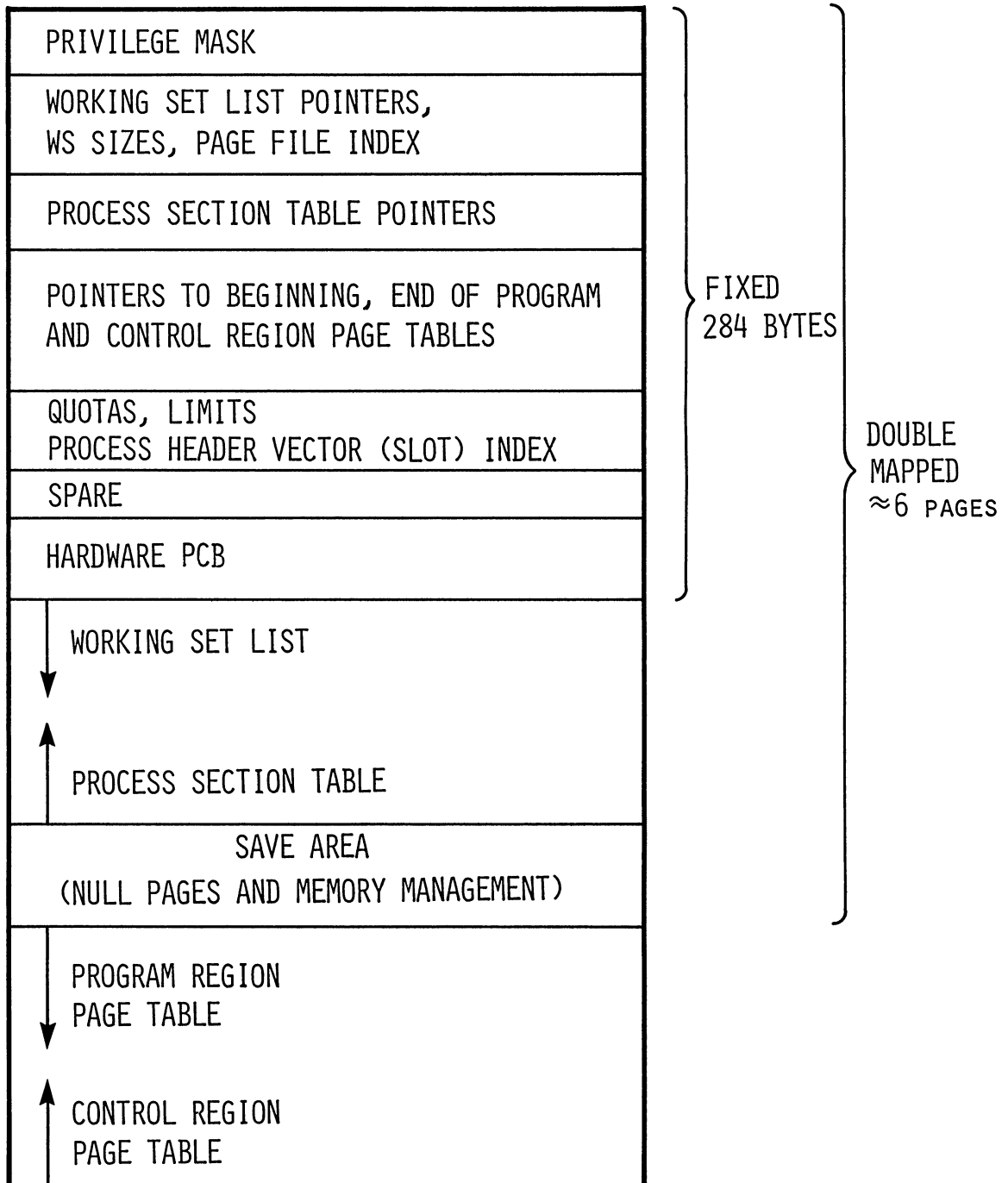
PCB\$L SQFL			
DCB\$L SQBL			
PCB\$B PRI	PCB\$B TYPE	PSBC\$W SIZE	
PCB\$W MTXCNT		PCB\$B ASTEN	PCB\$B ASTACT
PCB\$L ASTQFL			
PCB\$L ASTQBL			
PCB\$L PHYPCB			
PCB\$L OWNER			
PCB\$L UIC			
PCB\$L WSSWP			
PCB\$L STS			
PCB\$L WTIME			
PCB\$B PRIB	PCB\$B WEFC	PCB\$W STATE	
PCB\$W TMBU		PCB\$W APTCNT	
PCB\$W PPGCNT		PCB\$W GPGCNT	
PCB\$W BIOCNT		PCB\$W ASCNT	
PCB\$W BYTCNT		PCB\$W BIOLM	
PCB\$W DIOLM		PCB\$W DIOCNT	
PCB\$W TQCNT		PCB\$W FILCNT	
PCB\$L EFWM/PCB\$L PQB			
PCB\$L EFC			
PCB\$L EFCU			
PCB\$L EFC2P			
PCB\$L EFC3P			
PCB\$L PID			
PCB\$L PHD			
PCB\$W PRCCNT		PCB\$W BYTLM	
PCB\$T LNAME			

124 BYTES

A5

TK-3517

Figure 1-5. Software Process Control Block



A6

TK-1239

Figure 1-6. Process Header


```

0000 31      .SBTTL  SCH$RESCHED RESCHEDULING INTERRUPT HANDLER
0000 32      ;++
0000 33      ; SCH$RESCHED - RESCHEDULING INTERRUPT HANDLER
0000 34      ;
0000 35      ; THIS ROUTINE IS ENTERED VIA THE IPL 3 RESCHEDULING INTERRUPT.
0000 36      ; THE VECTOR FOR THIS INTERRUPT IS CODED TO CAUSE EXECUTION
0000 37      ; ON THE KERNEL STACK.
0000 38      ;
0000 39      ; ENVIRONMENT:
0000 40      ; IPL=3 MODE=KERNEL IS=0
0000 41      ; INPUT:
0000 42      ; 00(SP)=PC AT RESCHEDULE INTERRUPT
0000 43      ; 04(SP)=PSL AT INTERRUPT.
0000 44      ;--
0000 45      .ALIGN  LONG
0000 46      .SCH$RESCHED:  ;RESCHEDULE INTERRUPT HANDLER
0000 47      SETIPL #IPL$ _SYNCH ;SYNCHRONIZE SCHEDULER WITH EVENT REPORTING
0000 48      [SVPCTX] ;SAVE CONTEXT OF PROCESS
0000 49      MOVL W$SCH$GL_CURPCB,R1 ;GET ADDRESS OF CURRENT PCB
0000 50      MOVZBL PCB$B_PRI(R1),R2 ;CURRENT PRIORITY
0000 51      BBSS R2,W$SCH$GL_COMQ$ ,10$ ;MARK QUEUE NON-EMPTY
0000 52      0000*CF 07 0004 48      MOVL W$SCH$GL_CURPCB,R1
0000 53      0000*CF 52 E2 0000 49      MOVZBL PCB$B_PRI(R1),R2
0000 54      2C A1 0C 00 0013 51 10$: MOVL W$SCH$GL_COMQ$,10$ ;MARK QUEUE NON-EMPTY
0000 55      0000*CF 42 7E 0017 52      MOVAQ W$SCH$AQ_COMH[R2],R3 ;COMPUTE ADDRESS OF QUEUE
0000 56      93 61 0E 001D 53      INSQUE (R1),0(R3)+ ;INSERT AT TAIL OF QUEUE
0000 57      0020 54
0000 58      0020 55 ;+
0000 59      0020 56 ; SCH$SCHED - SCHEDULE NEW PROCESS FOR EXECUTION
0000 60      0020 57 ;
0000 61      0020 58 ; THIS ROUTINE SELECTS THE HIGHEST PRIORITY EXECUTABLE PROCESS
0000 62      0020 59 ; AND PLACES IT IN EXECUTION.
0000 63      0020 60 ;=
0000 64      0020 61 SCH$SCHED: ;SCHEDULE FOR EXECUTION
0000 65      0020 62 SETIPL #IPL$ _SYNCH ;SYNCHRONIZE SCHEDULER WITH EVENT REPORTING
0000 66      0020 63 FFS #0,#32,W$SCH$GL_COMQ$,R2 ;FIND FIRST FULL STATE
0000 67      52 20 00 EA 0023 64      BEQL SCH$IDLE ;NO EXECUTABLE PROCESS??
0000 68      0000*CF 3D 13 002A 65      MOVAQ W$SCH$AQ_COMH[R2],R3 ;COMPUTE QUEUE HEAD ADDRESS
0000 69      53 0000*CF 42 7E 002C 66      REMQUE 0(R3)+,R4 ;GET HEAD OF QUEUE
0000 70      54 93 0F 0032 67      BVS QEMPTY ;BR IF QUEUE WAS EMPTY (BUG CHECK)
0000 71      3C 1D 0035 68      BNEQ 20$ ;QUEUE NOT EMPTY
0000 72      06 12 0037 69      BBCC R2,W$SCH$GL_COMQ$,20$ ;SET QUEUE EMPTY
0000 73      0000*CF 52 E5 0039 70 20$:
0000 74      00 003E 71
0000 75      003F 72 20$: CMPB #DYN$C_PCB,PCB$B_TYPE(R4) ;MUST BE A PROCESS CONTROL BLOCK
0000 76      0A A4 0C 91 003F 73      BNEQ QEMPTY ;OTHERWISE FATAL ERROR
0000 77      2E 12 0043 74      MOVL W$SCH$GL_CURPCB ;NOTE CURRENT PCB LOC
0000 78      0000*CF 54 D0 0049 75      CMPB PCB$B_PRI(R4),PCB$B_PRI(R4) ;CHECK FOR BASE
0000 79      0B A4 2F A4 91 004E 76      BEQL 30$ ;YES, DONT FLOAT PRIORITY
0000 80      08 13 0053 77      BBC #4,PCB$B_PRI(R4),30$ ;DONT FLOAT REAL TIME PRIORITY
0000 81      08 A4 04 E1 0055 78      INCB PCB$B_PRI(R4) ;MOVE TOWARD BASE Prio
0000 82      08 A4 00 005D 79      MOVB PCB$B_PRI(R4),W$SCH$GB_PRI ;SET GLOBAL PRIORITY
0000 83      10 18 A4 0A 0063 80 30$: MTPR PCB$B_PHYPCB(R4),#PR$ _PCBB ;SET PCB BASE PHYS ADDR
0000 84      06 0067 81      LDPCTX ;RESTORE CONTEXT
0000 85      02 0068 82      REI ;NORMAL RETURN

```

A7

TK-3851

Figure 1-7. Rescheduling Interrupt Service Routine

BLANK

APPENDIX A – CONTENTS OF THE SOFTWARE PCB

Offset	Function
PCB\$W_APTCNT	Count of pages used for active page tables. Locates the process header within the working set swap image.
PCB\$B_ASTACT	AST active bits that denote access modes with currently active ASTs inhibiting the delivery of subsequent ASTs to that mode.
PCB\$W_ASTCNT	Number of ASTs pending for this process.
PCB\$B_ASTEN	AST enable bits for each access mode. These bits permit program control of AST delivery to that mode.
PCB\$L_ASTQBL	Tail or backward link of AST queue.
PCB\$L_ASTQFL	Head or forward link of AST queue.
PCB\$W_BIOCNT	Current number of buffered I/O operations in progress.
PCB\$W_BIOLM	Total number of concurrent buffered I/O operations allowed to the process.
PCB\$W_BYTCNT	Number of bytes that the process has allocated for buffered I/O operations.
PCB\$W_BYTLM	Total number of bytes that the process is allowed to have in use for buffered I/O operations.
PCB\$W_DIOCNT	Current number of direct I/O operations in progress. Direct I/O operations cause the pages containing the buffer to be locked in memory while the I/O operation is performed.
PCB\$W_DIOLM	Total number of concurrent direct I/O operations allowed to the process.
PCB\$L_EFCS	Process local event flag cluster number for system use. Contains the 32 event flags for cluster 0.
PCB\$L_EFCU	Process local event flag cluster number 1 for process use. Contains the 32 event flags for cluster 1.

Offset	Function
PCB\$\$_EFC2P	Pointer to common event block assigned to cluster number 2 within this process. Contains zero if not assigned.
PCB\$\$_EFC3P	Pointer to common event block assigned to cluster number 3 within this process. Contains zero if not assigned.
PCB\$\$_EFWM	Contains event flag wait mask when process is waiting for the setting of a local or common event flag. When a process is waiting for a mutual exclusion semaphore (mutex), it contains the mutex address. When a process is waiting for a dynamic resource, it contains an integer identifying the resource.
PCB\$\$_FILCNT	Number of files that the process has open.
PCB\$\$_GPGCNT	Count of global pages currently in the working set.
PCB\$\$_LNAME	Text string giving the logical name for the process. Up to 15 characters plus 1 byte count.
PCB\$\$_MTXCNT	Number of mutexes owned by this process.
PCB\$\$_OWNER	Process identification of subprocess owner. A zero value indicates a detached process.
PCB\$\$_PHD	Address of process header slot assigned to this process while it is a balance set member. A zero value indicates none assigned.
PCB\$\$_PHYPCB	Physical address of hardware PCB. The scheduler uses this value to initialize PCB prior to executing a load process context instruction.
PCB\$\$_PID	Process identification. A unique identifier for this process that is used to mark all resources it obtains.
PCB\$\$_PQB	Indication of the quota that a process attempted to exceed. The process waits until it can proceed without exceeding the indicated quota.
PCB\$\$_PPGCNT	Count of process private pages currently in the working set.

Offset	Function
PCB\$W_PRCNT	Number of subprocesses created by this process; that is, the number that currently exists.
PCB\$B_PRI	Current priority for process which selects proper scheduling subqueue for executable processes that are in or out of the balance set.
PCB\$B_PRIB	Base priority from which current priority is derived.
PCB\$W_SIZE	Size of PCB in bytes.
PCB\$L_SQBL	Backward link for state queue for process' current state.
PCB\$L_SQFL	Forward link for current state queue.
PCB\$W_STATE	State number representing the current state of the process.
PCB\$L_STS	Vector of 32 status flags for the process.
PCB\$W_TMBU	Termination mailbox unit number. Identifies a mailbox to which a termination message should be sent when the process terminates.
PCB\$W_TWCNT	Count of active timer queue entries made by this process.
PCB\$B_TYPE	Structure type code for PCB.
PCB\$L_UIC	User identification code for process. It is used to enforce a variety of protection checks.
PCB\$B_WEFC	Contains number of the cluster containing event flags for which the process is waiting when process is in an event flag wait state.
PCB\$L_WSSWP	Locates working set swap image for process. A zero value when the process is first created causes an inswap of the shell process image.
PCB\$L_WTIME	Contains time at start of wait. Used to compute elapsed wait time and response time.

BLANK

APPENDIX B – FIXED PORTION OF PROCESS HEADER

Process Privilege Mask (PHD\$Q_PRIVMSK)	
Authorized Working Set Size (PHD\$W_WSAUTH)	1st Working Set List Entry (PHD\$W_WSLIST)
1st Dynamic WS List Entry (PHD\$W_WSDYN)	1st Locked Working Set List Entry (PHD\$W_WSLOCK)
Last WS List Entry in List (PHD\$W_WSLAST)	Last WS List Entry Replaced (PHD\$W_WSNEXT)
Reference Fault Count (PHD\$L_REFERFLT)	
Default Working Set Size (PHD\$W_DFWSCNT)	Quota on Working Set Size (PHD\$W_WSQUOTA)
Pagfil Indx Byte (PHD\$B_PAGFIL)	Paging File Index, Long Word Reference (PHD\$L_PAGFIL)
Byte Offset to First Longword Beyond PST from Beginning of PHD (PHD\$L_PSTBASOFF)	
Head of Free PSTE List (PHD\$W_PSTFREE)	Address of Last PSTE Allocated (PHD\$W_PSTLAST)
1st Free Virtual Address at End of P0 Space (PHD\$L_FREP0VA)	
Count of Free PTEs Between the Ends of the P0 & P1 Page Tables (PHD\$L_FREPTECNT)	
1st Free Virtual Address at End of P1 Space (PHD\$L_FREP1VA)	
Count of Page File Pages that May Still Be Created (PHD\$L_PGFLCNT)	

Control Flags Word (PHD\$W_FLAGS)	Page Table Cluster Factor (PHD\$B_PGTBPFC)	Default Page Fault Cluster (PHD\$B)DFPFC)
Accumulated CPU Time Charged (PHD\$L_CPUTIM)		
Subprocess Quota (PHD\$W_PRCLM)	Accumulated CPU Time Since Last Quantum Overflow (PHD\$W_QUANT)	
Process Header Vector Index (PHD\$W_PHVINDEK)	AST Limit (PHD\$W_ASTLM)	
Pointer to WSL Index Save Area/ Longword Offset to Top of PST (PHD\$W_WLSX)/(PHD\$W_PSTBASMAX)	Pointer to Backup Address Vector for Process Header pages (PHD\$W_BAK)	
Count of Page Faults (PHD\$L_PAGEFLTS)		
Direct I/O Count (PHD\$L_DIOCNT)		
Buffered I/O Count (PHD\$L_BIOCNT)		
Limit on CPU Time for Process (PHD\$L_CPULIM)		
Maximum Virtual Page Count (PHD\$L_PGFLQUOTA)		
Timer Queue Entry List (PHD\$W_TQLM)	Open File Limit (PHD\$W_FILLM)	
Byte Offset to Byte Array of Counts of Locked WSLEs in this PGTBL (PHD\$L_PTWSLELCK)		
Byte Offset to Byte Array of Counts of Valid WSLEs in this PGTBL (PHD\$L_PTWSLEVAL)		
Count of Page Tables Containing One or More Valid WSLE (PHD\$W_PTCNTVAL)	Count of Page Tables Containing One of More Locked WSLE (PHD\$W_PTCNTLCK)	
Maximum Count of Page Tables Which Have Nonzero PTEs (PHD\$W_PTCNTMAX)	Count of Active Page Tables (PHD\$W_PTCNTACT)	

Extra Dynamic Working Set List Entries Above Required WS Fluid (PHD\$W_EXTDYNWS)	Guaranteed Number of Fluid Working Set Pages (PHD\$W_WSFLUID)
Hardware PCB / Kernel Stack (PHD\$L_PCB) / (PHD\$L_KSP)	
Executive Stack Pointer (PHD\$L_ESP)	
Supervisor Stack Pointer (PHD\$L_SSP)	
User Stack Pointer (PHD\$L_USP)	
Register 0 (PHD\$L_R0)	
Register 1 (PHD\$L_R1)	
Register 2 (PHD\$L_R2)	
Register 3 (PHD\$L_R3)	
Register 4 (PHD\$L_R4)	
Register 5 (PHD\$L_R5)	
Register 6 (PHD\$L_R6)	
Register 8 (PHD\$L_R8)	
Register 9 (PHD\$L_R9)	
Register 10 (PHD\$L_R10)	
Register 11 (PHD\$L_R11)	

Argument Pointer (PHD\$ <u>L</u> _R12)	
Frame Pointer (PHD\$ <u>L</u> _R13)	
Program Counter (PHD\$ <u>L</u> _PC)	
Process Status Longword (PHD\$ <u>L</u> _PSL)	
P0 Base Register (PHD\$ <u>L</u> _P0BR)	
AST Level Subfld (PHD\$ <u>B</u> _ASTLVL)	P0 Length Register (PHD\$ <u>L</u> _P0LRASTL)
P1 Base Register (PHD\$ <u>L</u> _P1BR)	
P1 Length Register (PHD\$ <u>L</u> _P1LR)	
Spare Word	Count of Empty Working Set Pages (PHD\$ <u>W</u> _EMPTPG)
Spare	
Spare	
Spare	
Spare	
Spare	
Spare	
Spare	
Spare	
Spare	
Spare	
Spare	

Spare
Spare
Spare
Spare
PHD\$C_LENGTH - Length of Fixed Portion of Header
First Working Set List Entry (PHD\$L_WSL)

BLANK

APPENDIX C – PROCESS ENVIRONMENT

Introduction

The concept of the process is a fundamental principle of VAX/VMS. You have learned that a process is an environment in which an image (the executable form of a program) executes. The process provides three major resources to an image:

- Hardware context - the contents of the general purpose and process-specific processor registers
- Software context - accounting data, privileges, priorities, etc.
- Address space - a range of virtual addresses and the means of translating them into physical ones.

VAX/VMS manages these resources on behalf of the user by manipulating several data structures. This appendix introduces these data structures. Each of them will be examined in more detail as its interaction with the various VAX/VMS components is explored in subsequent modules of this course. For further clarification of image, read User Process, on page 2 of the Software Handbook (1978-1979).

Hardware Context

The hardware context of a process is a combination of the contents of the general purpose registers and the processor registers that are process specific. When a process is the current, executing process, the hardware context resides in the hardware registers. When a process is not executing, its hardware context is stored in its hardware process control block (PCB). The hardware PCB is part of a larger structure called the process header.

The following list of abbreviations and their meanings relate to hardware context.

Abbreviations	Meaning
KSP	Kernel Stack Pointer
ESP	Executive Stack Pointer
SSP	Supervisor Stack Pointer
USP	User Stack Pointer
AP	Argument Pointer (R12)
FP	Frame Pointer (R13)
PC	Program Counter (R15)
PSL	Processor Status Longword
P0BR,P0LR	P0 (Program Region) Base and Length Registers
P1BR,P1LR	P1 (Control Region) Base and Length Registers
ASTLVL	Asynchronous System Trap Level
PME	Performance Monitor Enable

Software Context

The software context of a process includes such information as the process state and priority of the process, the resources allocated to the process, and information about asynchronous system traps (ASTs) associated with the process. In addition, there are pointers to other structures, including:

- State queue of other PCBs
- Queue of pending ASTs for this process
- Hardware PCB (a physical address)
- Common event flag clusters
- Process headers
- Slot in the swap file assigned to this process

The software PCB resides in system address space and is not swapped.

Virtual Address Space

The address space of the process is divided into two regions: a program region (P0 space) and a control region (P1 space). Each region may be up to 1 gigabyte long. This range of addresses is managed by the process header, a data structure in system space that is both paged and swapped. It has been already noted that the hardware PCB is part of the process header. Other components include:

1. The page tables for the program and control regions which provide mapping from virtual addresses to physical locations;
2. The working set list of pages/page frames which describes the collection of pages whose address translations are valid;
3. Privilege mask which is used to determine the authority of a process to perform potentially powerful operations; and
4. Pointers to image file locations for sections of the current image within the process which assist in paging in pure (unchanged) copies of requested pages.

Locations in the process header (except for the page tables) also may be accessed through a range of addresses in the control region (P1 space). These locations are doubly mapped. The process header is included in the working set of the process.

Program Region

The program region (P0 space) of the address space of a process contains the program code and data areas specified by the user. Virtual addresses range from 00000000 to 3FFFFFFF. Although the user may exercise considerable control over how an image is built, the linker constructs an image in sections as shown on the opposite page. Each image section has unique attributes. The specific allocation of space is based upon the following concerns.

1. Previously linked shared non-PIC code must be placed at its associated base address.
2. The remaining nonposition independent coding (PIC) portions of the image are allocated next to facilitate their mapping.
3. PIC portions can be shared at any location in the program region.

4. Nonshared PIC code can be placed wherever it fits.
5. Page table entries are required to reserve unallocated space. Therefore, the address space is packed as tightly as possible to avoid page table entries which are unused.

The program region is paged, swapped and included in the working set of the process.

A "no access" or "null" page contains no data, is inaccessible to all access modes, and requires only one longword of storage (a page table entry).

Control Region

The control region (P1 space) of the address space of process contains several components, including:

1. The four per process stacks
2. Data structures associated with image and process I/O
3. Impure data areas used by the command language interpreter (CLI)
4. Virtual addresses mapped to the shared image of the CLI
5. Areas for use by the image activator and the symbolic debugger
6. A "window" into the first portion of the process header

The control region also is paged and swapped, and it, too, counts against the working set of the process.

Virtual addresses range from 40000000 to 7FFFFFFF.

System Region

The remaining area of interest is the system region (virtual addresses from 80000000 to BFFFFFFF). In this region are operating system routines, VAX-11 RMS, space for operating system data structures (including I/O, memory management, software PCBs, and process headers), as well as structures for handling exceptions and interrupts (interrupt stack, software vectors, and system control block). Thus, the system region contains the VAX/VMS operating system components and the resources sharable by all users.

MODULE TEST

1. The hardware context of a process is a combination of the contents of the general purpose registers and the process registers that are process specific.

Indicate by placing an X to the left of the registers that are part of the hardware context.

Note: Not all hardware context registers are listed.

- ☐ System control block base register
- ☐ Frame pointer register
- ☐ Interrupt priority level register
- ☐ System length register
- ☐ Argument pointer register
- ☐ Processor status longword register
- ☐ Software interrupt request register
- ☐ Asynchronous system trap level register
- ☐ Interrupt stack pointer register
- ☐ P0 base register
- ☐ Process control block base register
- ☐ SBI fault/status register
- ☐ P1 length register
- ☐ System length register
- ☐ Kernel stack pointer register

2. Circle the letter of the statement that describes a characteristic of the software context. (Software context is also called software PCB.)
 - a. The software PCB resides in virtual address space and can be swapped.
 - b. The software PCB resides in system address space and can be swapped.
 - c. The software PCB resides in virtual address space and is not swapped.
 - d. The software PCB resides in system address space and is not swapped.

MODULE TEST

3. For each feature or resource associated with or used by a process:
- Name the data structure or code that implements or controls it
 - Check the region in which the data structure or code resides
 - Check whether the data structure or code is paged or swapped.

Use the chart provided. The first was done as an example.

Feature	Data Structure	Region			Swapped	Paged
		Program	Control	System		
Process priority	Software PCB		X	X	X	X
User stack						
Page tables						
Process privilege mask						
Kernel stack						
Interrupt stack						
Contents of GPRs when process is not active						

ADDRESS TRANSLATION

2

INTRODUCTION

While diagnosing certain hardware failures, the student may use various troubleshooting aids, including:

1. PCS listing
2. Error log printout
3. Macrosource listing

Some of the required terminology the student needs to interpret will be covered within this module. Also, the student will be introduced to the following:

1. Address translation sequence
2. Types of exceptions that can occur during an address translation
3. Examples of how VMS uses the various fields within a page table entry.

OBJECTIVES

1. Name the two types of memory management faults and their respective vector locations within the system control block.
2. List the conditions that would create memory management exceptions.
3. Determine which address region will be accessed by a given virtual address.
4. Given a field contained within a page table entry, describe its function.

5. Given the contents of the following, convert a virtual address to a physical address.
- a. P0 base register
 - b. P0 length register
 - c. P1 base register
 - d. P1 length register
 - e. System base register
 - f. System length register
 - g. Physical addresses of all page table entries

SAMPLE TEST ITEM

Read the following and decide if the statement is true or false. Circle your response.

1. Virtual address space can be divided into two pieces by using bit 31 of the virtual address.

Statement: When bit 31 is set, the virtual address is a process virtual address found in process space.

TRUE

FALSE

RESOURCES

- 1. Introduction to VAX-11: Concepts, A/V Course, Memory Management Module
- 2. VAX-11/780 Hardware Handbook, Chapter 6
- 3. Translation Buffer, Cache and SBI Control Technical Description (VAX-11/780 Implementation), Sections 1.1 through 1.7, Section 1.9, Section 2.1

LECTURE OUTLINE

- I. Concepts
 - A. Protection
 - B. Mapping
 - C. Physical address space
 - II. Page Tables and Mapping Registers
 - A. Page tables
 - B. Page table entries
 - C. Base and length registers
 - III. Memory Management Exceptions
 - A. Access violation
 - 1. Protection violation
 - 2. Length violation
 - B. Translation not valid fault
 - IV. Address Translation
 - A. System virtual address translation
 - B. Process virtual address translation
 - 1. P0 translation
 - 2. P1 translation
 - C. Translation buffer
 - D. Address translation faults
- Appendix A Translation Buffer

BLANK

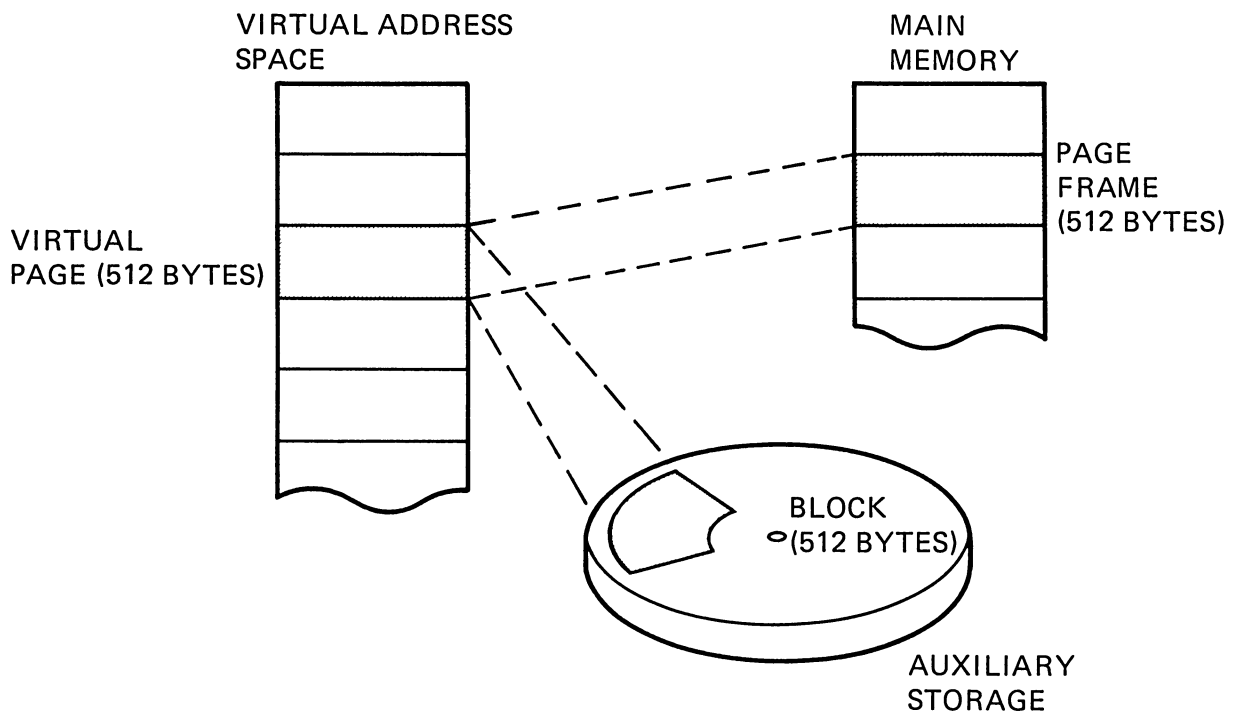
CONCEPTS

The implementation of a hardware memory management system to translate program virtual addresses provides an operating system with two valuable tools, memory protection and mapping.

Protection

By including memory protection checks into the address translation scheme, the operating system and user programs may prevent pages from being written into or modified (read-only) or from being accessed at all (no access).

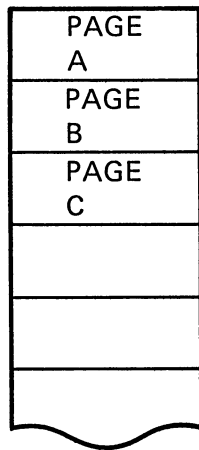
In the VAX-11/780, the unit of memory to which protection can be applied is the page (512 bytes). Each of the four access modes (kernel, executive, supervisor, user) can have one of three access rights to a page (no access, read-only, read-write).

Mapping

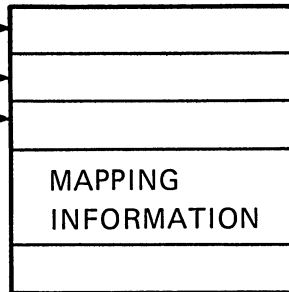
TK-3539

Figure 2-1. Virtual Addressing. Since virtual addresses on VAX-11/780 are 32-bits long, a program has a potential 4.3 gigabytes available to it. In general, only a small portion of virtual address space actually corresponds to pages in physical memory. Portions of virtual address space may instead be found on auxiliary storage which may include the original image file from which a program was run, the page file or a global image file.

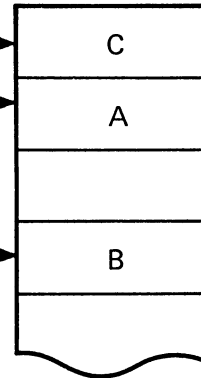
VIRTUAL ADDRESS SPACE
(4.3 BILLION BYTES)



PAGE
TABLE

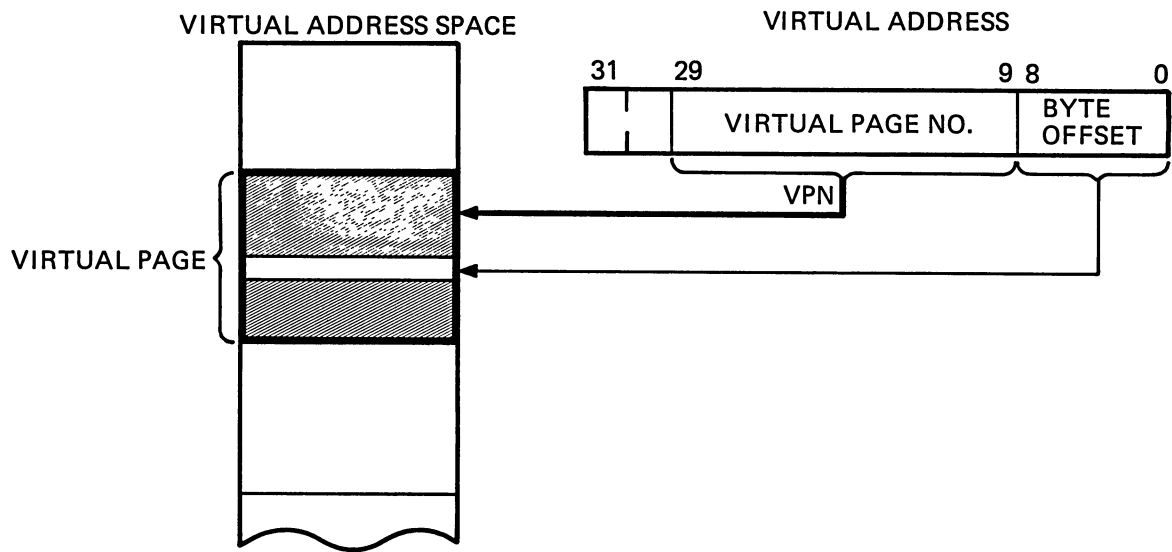


PHYSICAL ADDRESS SPACE
(1 BILLION BYTES)



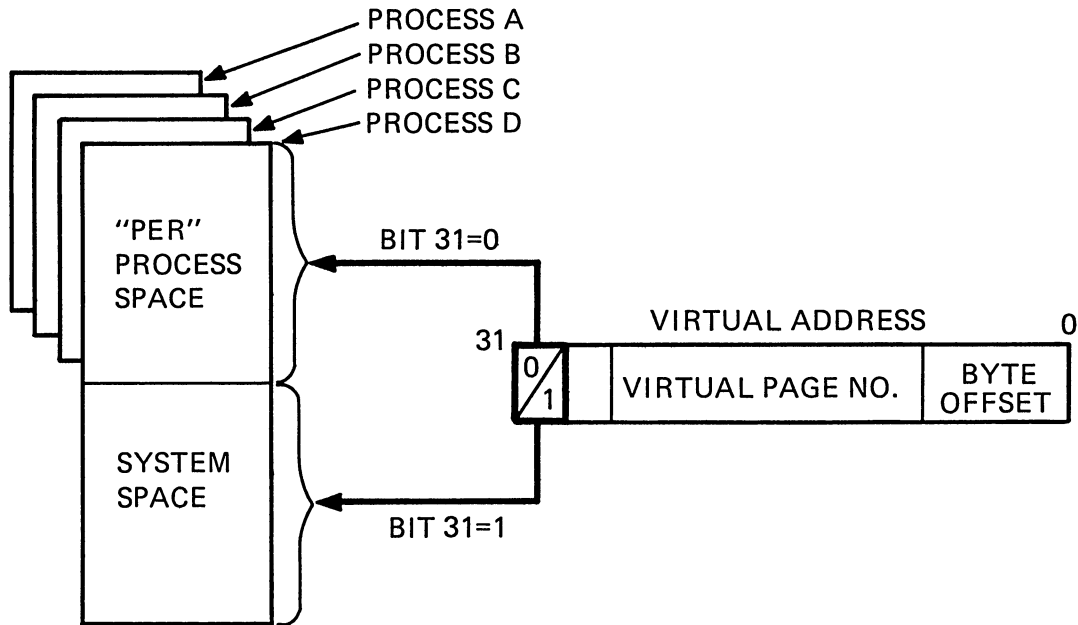
TK-3540

Figure 2-2. Address Translation. The mapping from virtual addresses to either physical addresses or auxiliary storage locations uses the virtual address and another data structure called page tables to correctly map each virtual page.



TK-3549

Figure 2-3. Virtual Address Space. Virtual address space can be thought of as consisting of a series of virtual pages which can be numbered using bits 29 through 9 of a given virtual address. Bits 8 through 0 specify which byte within the page is being addressed.



TK-3541

Figure 2-4. Virtual Address P0/P1 Space. Virtual address space can be divided into two pieces by using bit 31 of the virtual address. When bit 31 is set, ($VA \geq 80000000_{16}$) the address is called a system virtual address and references system space. System space is implicitly shared by all processes in the system. That is, two processes referring to the same system virtual location are referencing the same physical location.

When bit 31 is clear, the virtual address is a process virtual address found in process space or "per-process" space. Each process has its own process space and it is practically impossible for one process to refer to a process virtual address of another process.

Both process space and system space are further divided into two pieces determined by the setting of bit 30 in the virtual address.

1. $VA\langle 31:30 \rangle = 0$ Program Region

This portion of virtual address space is called P0 space or the program region. P0 space typically contains the code and data of an image being executed by the process.

2. $VA\langle 31:30 \rangle = 1$ Control Region

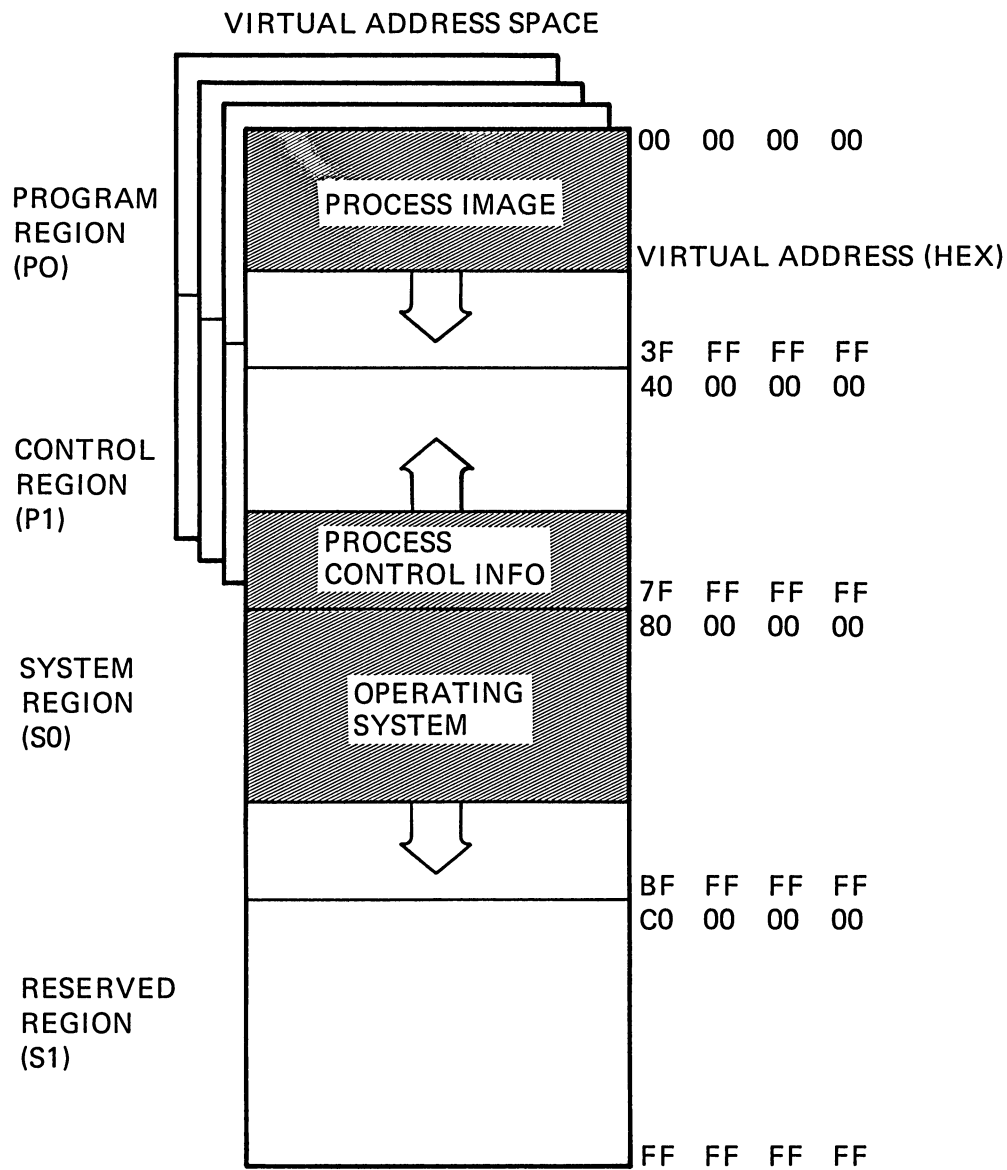
This portion of virtual address space is called P1 space or the control region. It contains such information as the four per-process stacks, a command language interpreter, DEBUG symbol table, process I/O data and so on.

3. $VA\langle 31:30 \rangle = 2$ System Region

This portion of virtual address space is called the system region. It contains the executive, device drivers and their associated data structures, RMS code and pure data, both the system and process page tables and other code and data that does not belong to any one process in the system.

4. $VA\langle 31:30 \rangle = 3$ Reserved

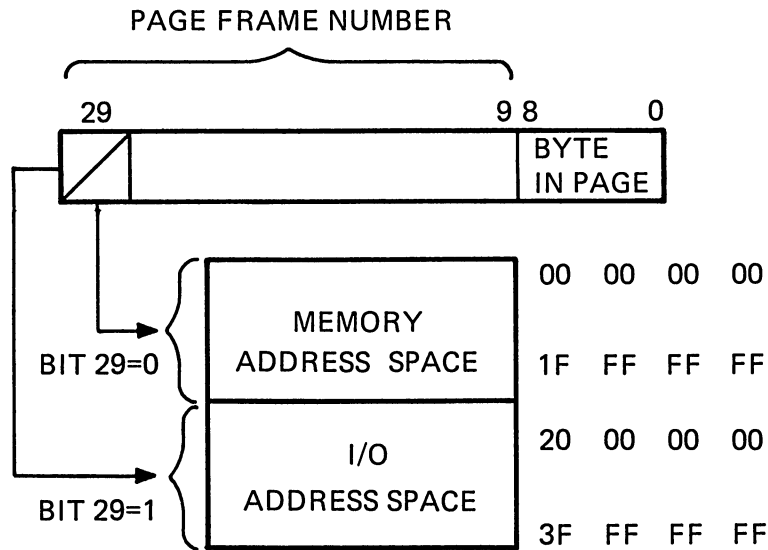
This portion of virtual address space is currently reserved. A reference to a virtual address in this range will cause a length violation.



TK-3542

Figure 2-5. Allocation of Virtual Address Space

Physical Address Space



TK-3544

Figure 2-6. Physical addresses are 30 bits long.

1. PA<29> = 0

These physical addresses reference main memory.

2. PA<29> = 1

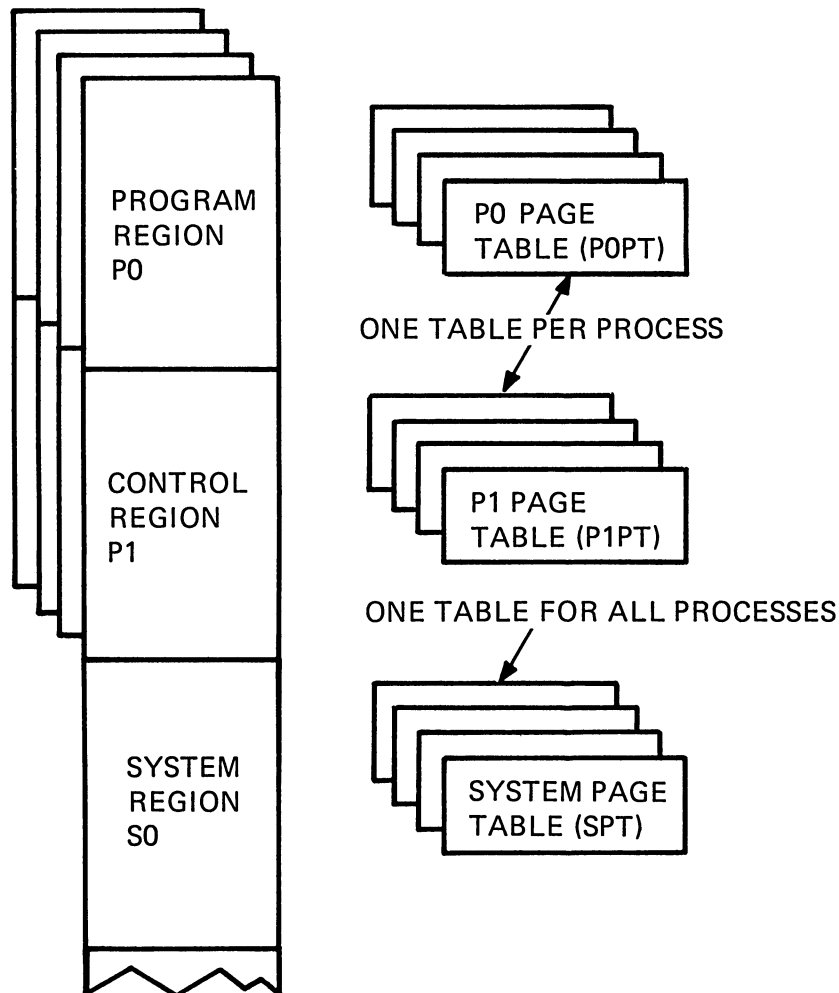
These physical addresses point to I/O addresses. These include:

- Memory controller registers
- MBA registers
- MASSBUS device registers
- UBA registers
- UNIBUS device registers
- UNIBUS address space

Address translation involves transforming a 32-bit virtual address into a 30-bit physical address. By setting the page table entry appropriately, any virtual address can point to either main memory or I/O addresses.

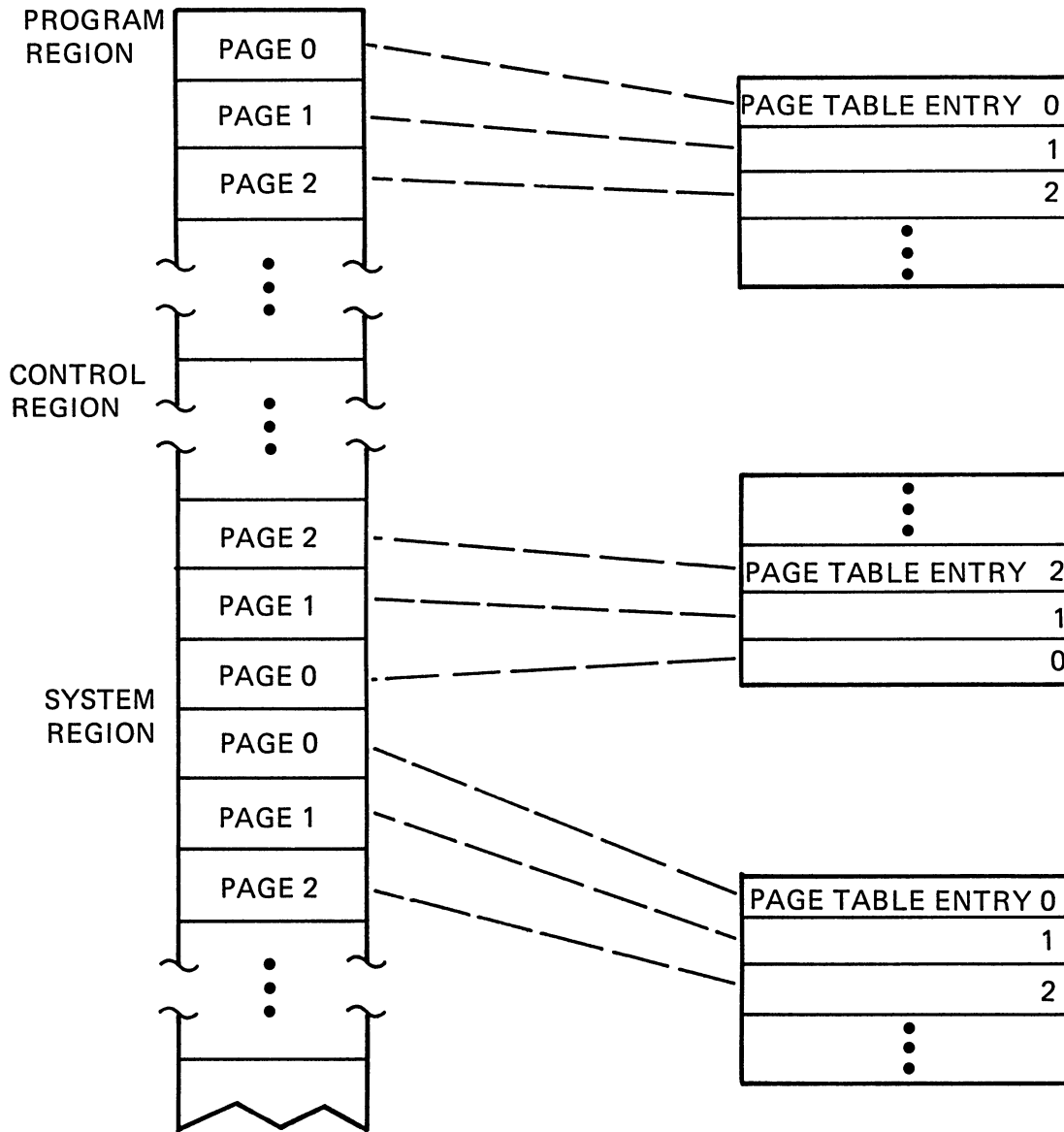
PAGE TABLES AND MAPPING REGISTERS

Page Tables



TK-3545

Figure 2-7. Process Page Tables (1). The operating system keeps track of the status and physical location of virtual pages in a set of data structures called page tables. Protection information is also found in these structures. There is a single page table used to map the system region called the system page table. In addition, each process has two page tables to map its process space, one each for P0 and P1 space.



TK-3543

Figure 2-8. Process Page Tables (2). Each page table contains a longword entry for each virtual page in the appropriate region. Notice that the P1PT, like P1 space, grows toward smaller addresses.

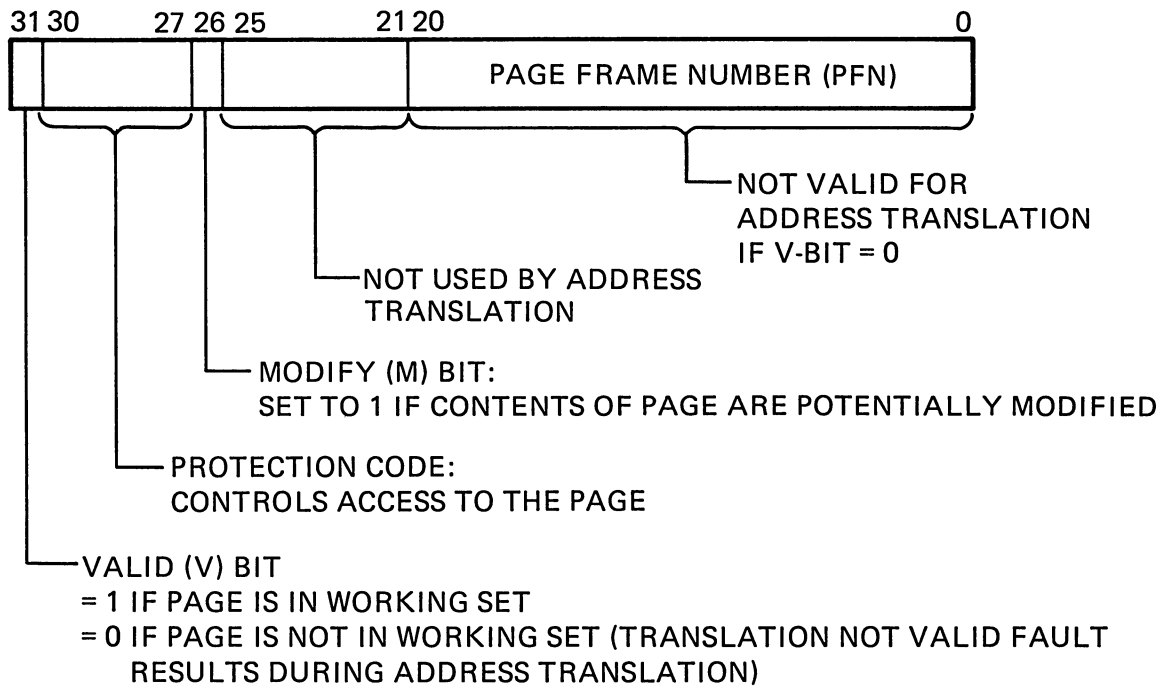
Page Table Entries

The system page table is built at initialization time and is located in contiguous pages of physical memory. Process page tables are set up at process creation and altered at image activation, image exit and in response to various system services. Process page tables are located in virtually contiguous pages of system space. That is, process page tables need not be physically contiguous. This design feature prevents a potentially serious fragmentation problem in physical memory.

The most significant bit in the PTE is called the valid bit. When this bit is set, the appropriate virtual page is in the working set. In addition, when a page is valid, the M-bit and the PFN field may be used by the VAX memory management. The modify bit indicates whether this page has been modified since last brought into the working set. The page frame name field indicates which physical page is mapped to by this virtual address.

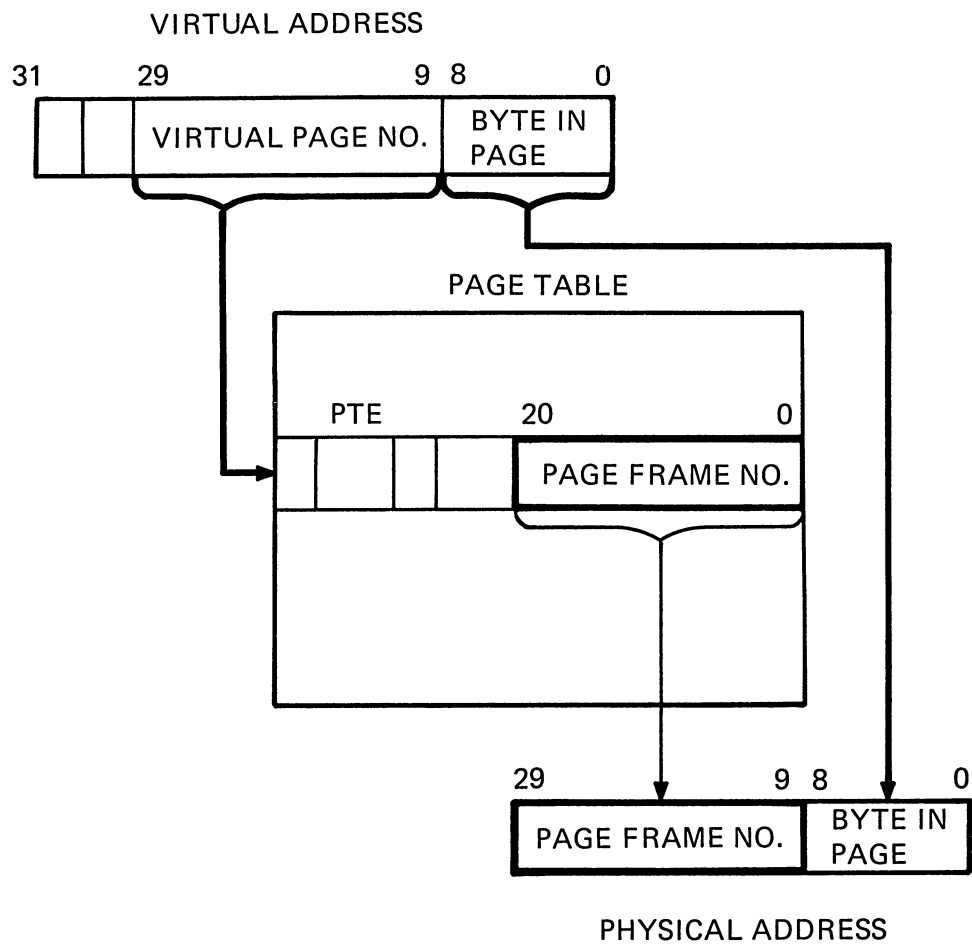
When the valid bit is clear, the operating system puts other information into the M-bit and PFN field to allow this page to be located on auxiliary storage or elsewhere.

The contents of the protection field are meaningful even when the valid bit is clear. This permits the address translation hardware to check protection before validity to avoid the costly overhead of faulting in a page to which the user was denied access rights.



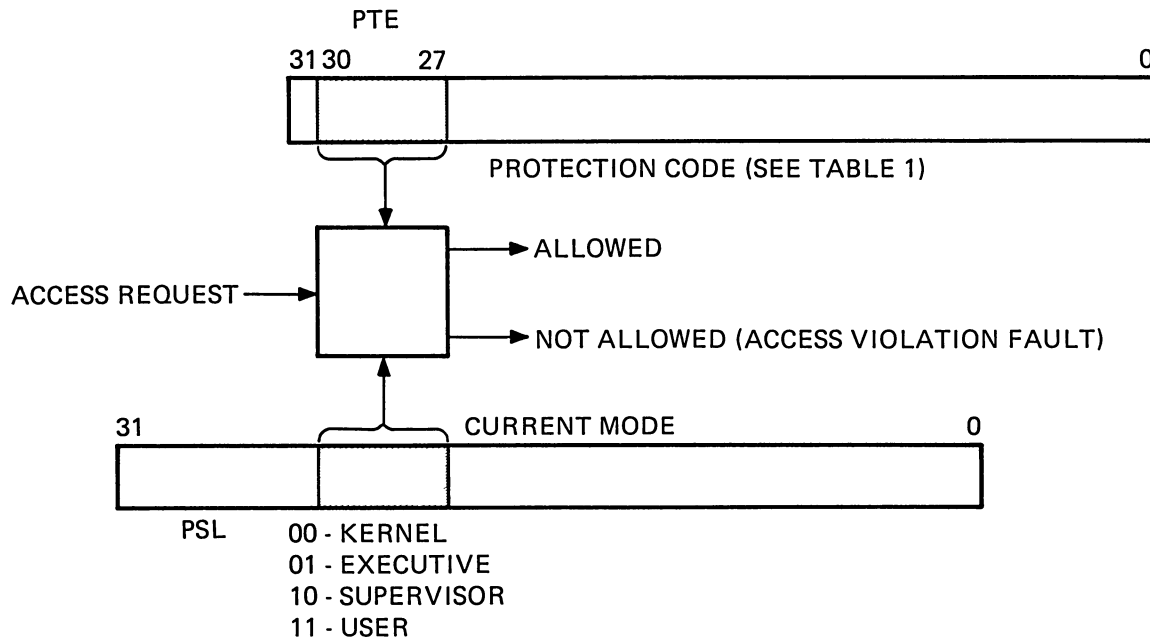
TK-3547

Figure 2-9. Fields in Page Table Entry Used by Address Translation Hardware



TK-3576

Figure 2-10. The Virtual Page Number Field of a virtual address selects a page table entry. If the valid bit is set, the PFN field of the PTE forms the upper 21 bits of a physical address. The byte offset field is simply carried through the translation.



TK-3548

Figure 2-11. Protection Field. The address translation uses the protection field, along with the currentmode field of the PSL and the intended access request, to determine whether or not this access will be allowed. If allowed, address translation continues. If not, an access violation fault occurs.

We mentioned in the first section that each of the four access modes could have any of three access rights to a given page. This produces 256 separate cases. However, two simple and reasonable assumptions reduce 256 cases to 15, which can be stored in four bits in the PTE.

1. If an access mode can write to a page, it can also read that page.
2. If an access mode has certain access rights to a page (read or read-write), then all more privileged access modes also have that access right.

Tables 2-1 below and Table 6-1 in the Hardware Handbook list the 15 possibilities in numerical order. The algorithm used by the hardware to determine if access is allowed is also found in the handbook and reproduced here.

Table 2-1. Use of Protection Codes for Access Control

PROTECTION CODE IN PTE	CURRENT ACCESS MODE			
	KERNEL	EXECUTIVE	SUPERVISOR	USER
0000	--	--	--	--
0001	UNPREDICTABLE		UNPREDICTABLE	--
0010				
0011				
0100	RW	RW	RW	RW
0101	RW	RW	--	--
0110	RW	R	--	--
0111	R	R	--	--
1000	RW	RW	RW	--
1001	RW	RW	R	--
1010	RW	R	R	--
1011	R	R	R	--
1100	RW	RW	RW	R
1101	RW	RW	R	R
1110	RW	R	R	R
1111	R	R	R	R

-- (NO ACCESS)
 R (READ-ONLY ACCESS)
 RW (READ AND WRITE ACCESS)

TK-3567

Access is allowed if:

(CODE NEQU 0)

AND

(CODE EQLU 4)

OR

(CM LSSU WM)

OR

[READ
 AND
 (CM LEQU RM)]

CM - Current mode field of PSL

RM - Left two bits of protection code

WM - One's complement of right two bits of protection code

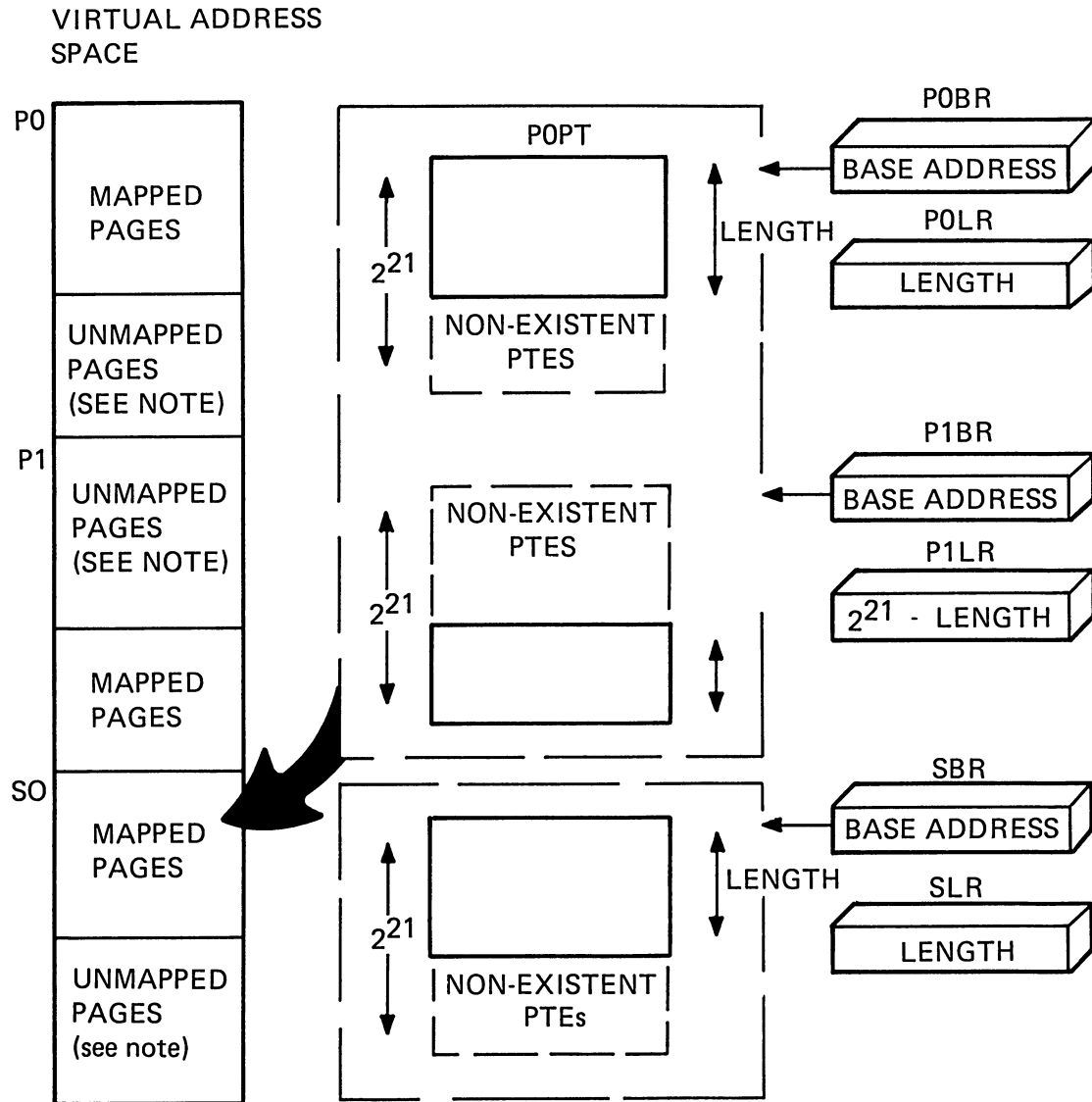
Table 2-2. The operating system may also reference (or more often modify) the protection field of the PTE. We list here the symbolics used by the code. These symbols are all defined by invoking the \$PTEDEF macro in the library SYS\$LIBRARY:LIB.MLB. The values listed are all shifted into the protection field. A similar set of symbols (using unshifted values) can be defined by invoking the macro \$PRTDEF, found in the default macro library SYS\$LIBRARY:STARLET.MLB. This macro defines symbols of the form PTE\$C_xxxx.

Symbol	Value (Binary) (@ 27)	Meaning
PTE\$C_NA	0000	No Access
PTE\$C_KR	0011	Kernel Read Only
PTE\$C_KW	0010	Kernel Write
PTE\$C_ER	0111	Exec Read Only
PTE\$C_EW	0101	Exec
PTE\$C_SR	1011	Super Read Only
PTE\$C_SW	1000	Super Write
PTE\$C_UR	1111	User Read Only
PTE\$C_UW	0100	User Write
PTE\$C_ERKW	0110	Exec Read Kernel Write
PTE\$C_SRKW	1010	Super Read Kernel Write
PTE\$C_SREW	1001	Super Read Exec Write
PTE\$C_URKW	1110	User Read Kernel Write
PTE\$C_UREW	1101	User Read Exec Write
PTE\$C_URSW	1100	User Read Super Write

Base and Length Registers

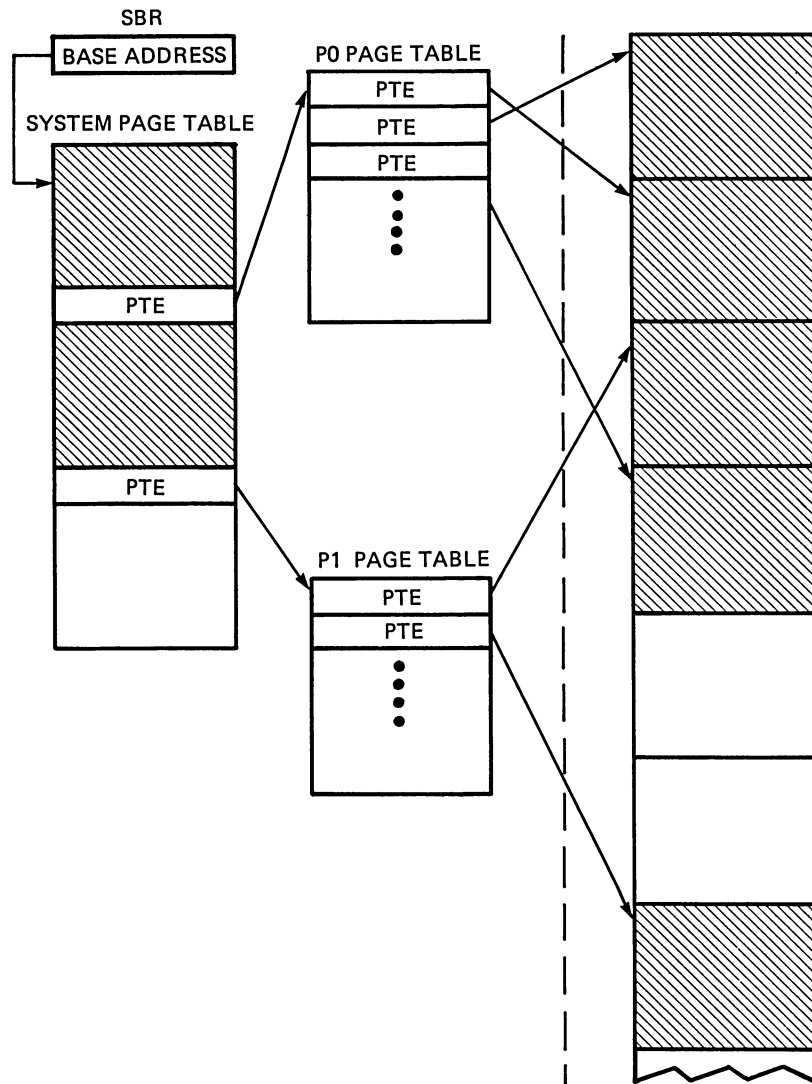
The information contained in the page tables is used not only by the operating system but also by the address translation hardware. To provide a simple means to locate the page tables, the processor contains three registers containing the base addresses of the three page tables. Note that the system base register contains a physical address while the P0 and P1 base registers contain virtual addresses. As we mentioned earlier, the system page table is physically (and also virtually) contiguous, while the P0 and P1 page tables are only virtually contiguous.

Note that the P1 base register points to what would be the PTE for the first (lowest addressed) page of the P1 region, and the length register actually counts the number of unmapped pages.



TK-3551

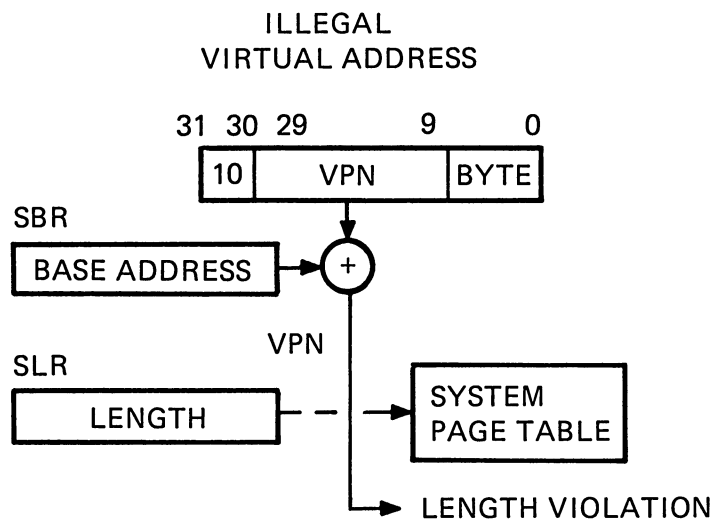
Figure 2-12. Base and Length Registers. The base registers locate their respective page tables. The length registers describe the size of each of the three regions.



TK-3546

Figure 2-13. Overall Relationship Between SPT, P0PT, P1PT and Process Pages. Since the P0 and P1 page tables are located in system virtual address space, they are mapped. That is, there are entries in the system page table which map process page tables from system virtual addresses to physical memory. The contents of these pages in turn map process virtual addresses to physical addresses.

Typically, a program uses only a small fraction of virtual address space available to it. In order for the operating system to avoid filling a large number of page table entries with "no access" protection, the processor maintains a second set of three registers which contain the number of PTEs in each page table containing meaningful information. (PLR actually contains the number of meaningless PTEs.) An attempt to reference a virtual page whose virtual page number is greater than or equal to (strictly less than for P1 addresses) the contents of the appropriate length register will result in a form of an access violation called a length violation.

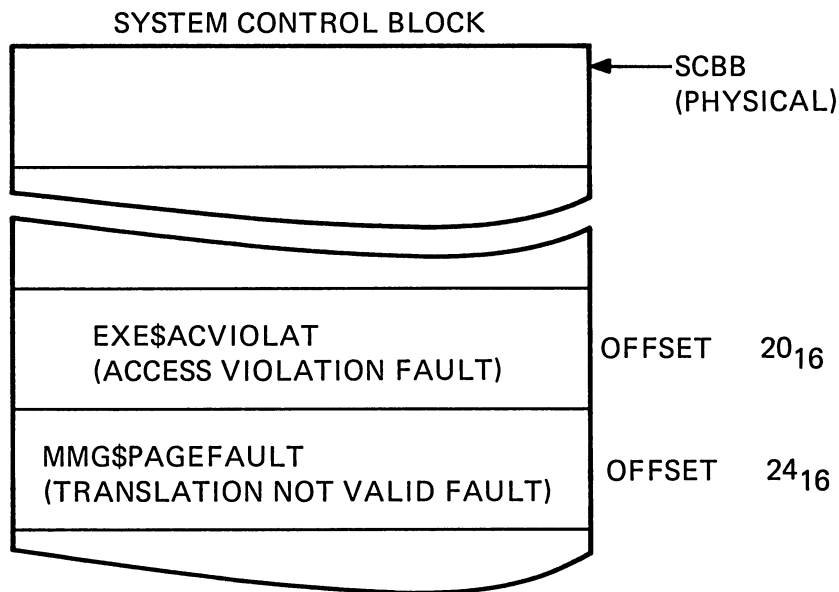


TK-3552

Figure 2-14. Example of a Length Violation. Recall that any reference to the so-called reserved region results in a length violation.

MEMORY MANAGEMENT EXCEPTIONS

During address translation, two different kinds of exception can occur: access violation and translation not-valid exceptions. Both forms of exception are faults. That is, the processor backs up the faulting instruction so that it can be restarted when (or if) the exception is resolved. Two adjacent longwords in the system control block are set up at initialization time to point to the routines which will service these exceptions. Both exceptions are handled on the kernel stack.



TK-3550

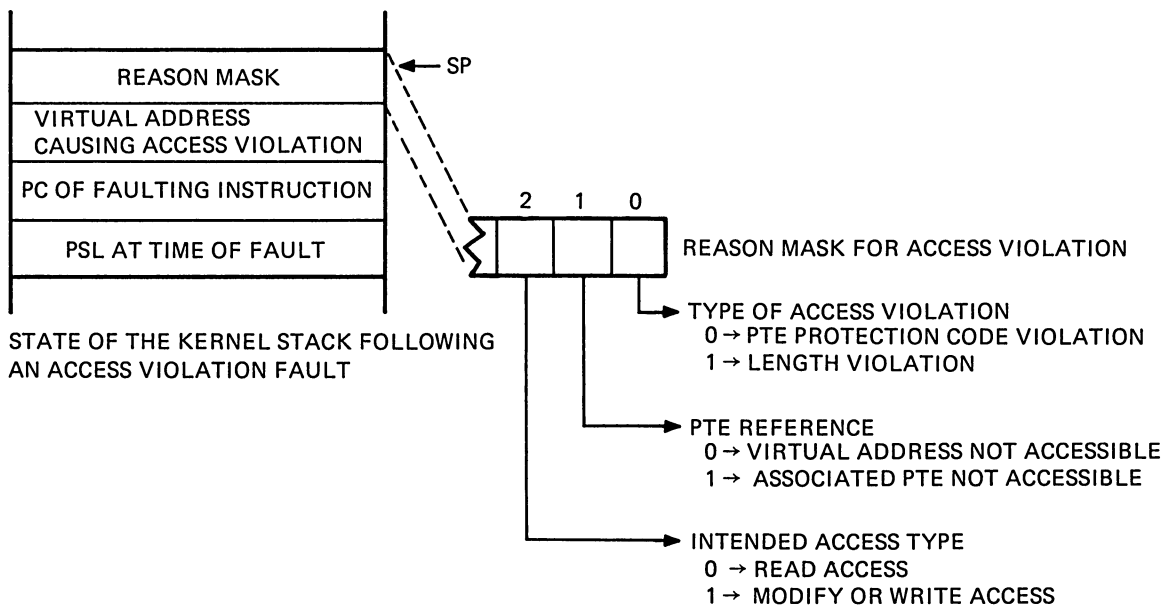
Figure 2-15. The System Control Block (SCB) contains the addresses of routines which will execute when exceptions and interrupts occur.

Access Violation

An access violation can occur in two different forms. A protection code violation occurs when the intended access request (read, modify, write) is not allowed for the current access mode. Recall that the protection code is found in bits <30:27> of the appropriate page table entry.

A length violation occurs when the virtual page number of the address to be translated is greater than or equal to the contents of the appropriate length register. (Because P1 space grows toward smaller addresses, the length violation fault occurs when $VA_{<29:9>}$ is strictly less than the contents of P1LR.)

When an access violation occurs, the faulting PSL and PC are pushed onto the kernel stack, followed by the virtual address which caused the access violation. Finally, a longword fully describing the access violation is pushed onto the stack. Note that bit <0> of the reason mask distinguishes length violations from protection code violations.



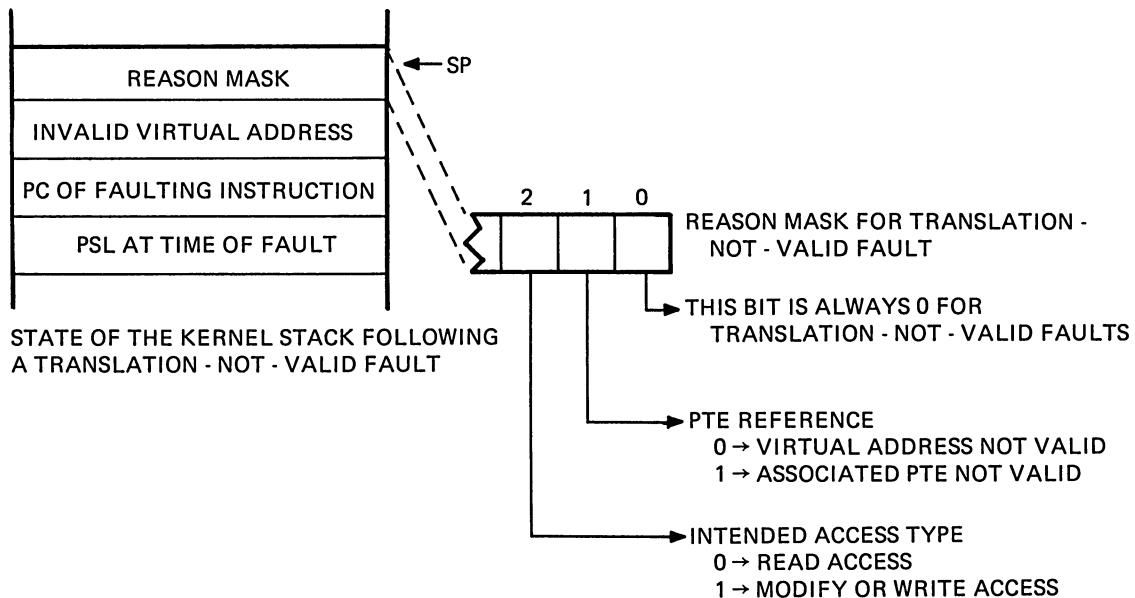
TK-3559

Figure 2-16. State of the Kernel Stack Following an Access Violation Fault

Translation-Not-Valid Fault

A translation-not-valid fault occurs when the valid bit (VA<31>) is clear. The faulting PSL and PC, followed by the invalid virtual address and reason mask, are pushed onto the kernel stack. Control is passed to an executive routine called the pager which will use the information in the invalid PTE to locate the page and add it to the working set of the requesting process.

Since process page tables are mapped (by SPT entries), address translation for process virtual addresses can incur page faults both in translating the system virtual address of the process page table entry and in translating the process virtual address itself. These two different cases are distinguished by bit <1> of the reason mask.



TK-3560

Figure 2-17. State of the Kernel Stack Following a Translation-Not-Valid Fault

NOTE

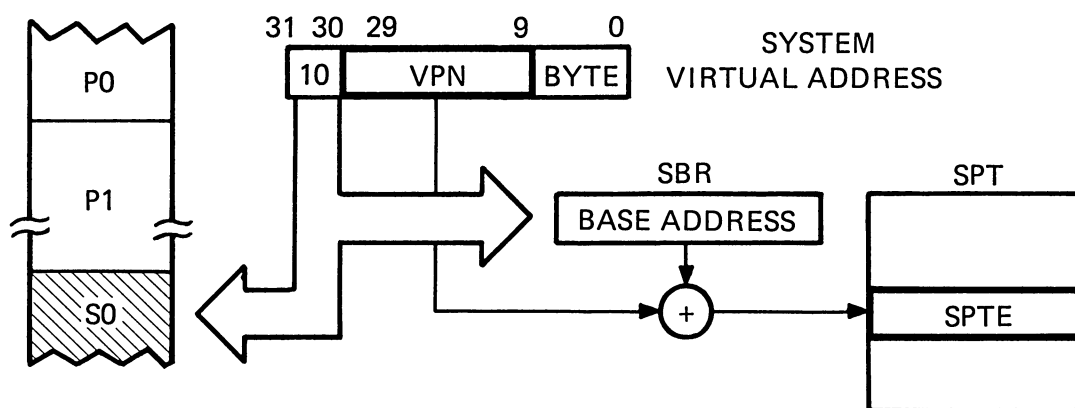
The address translation mechanism checks the protection code before it checks the valid bit. Thus, if a given address translation could cause both an access violation and a page fault, the access violation will be taken. This design avoids the overhead of faulting into a process working set a page which it is not allowed to access. A further discussion is found on page 108 of the VAX-11/780 Hardware Handbook.

ADDRESS TRANSLATION

Now we will explore the actual mechanics of address translation for both system virtual addresses and process virtual addresses. We will also point out the impact of the translation buffer and note the sequence of fault checking.

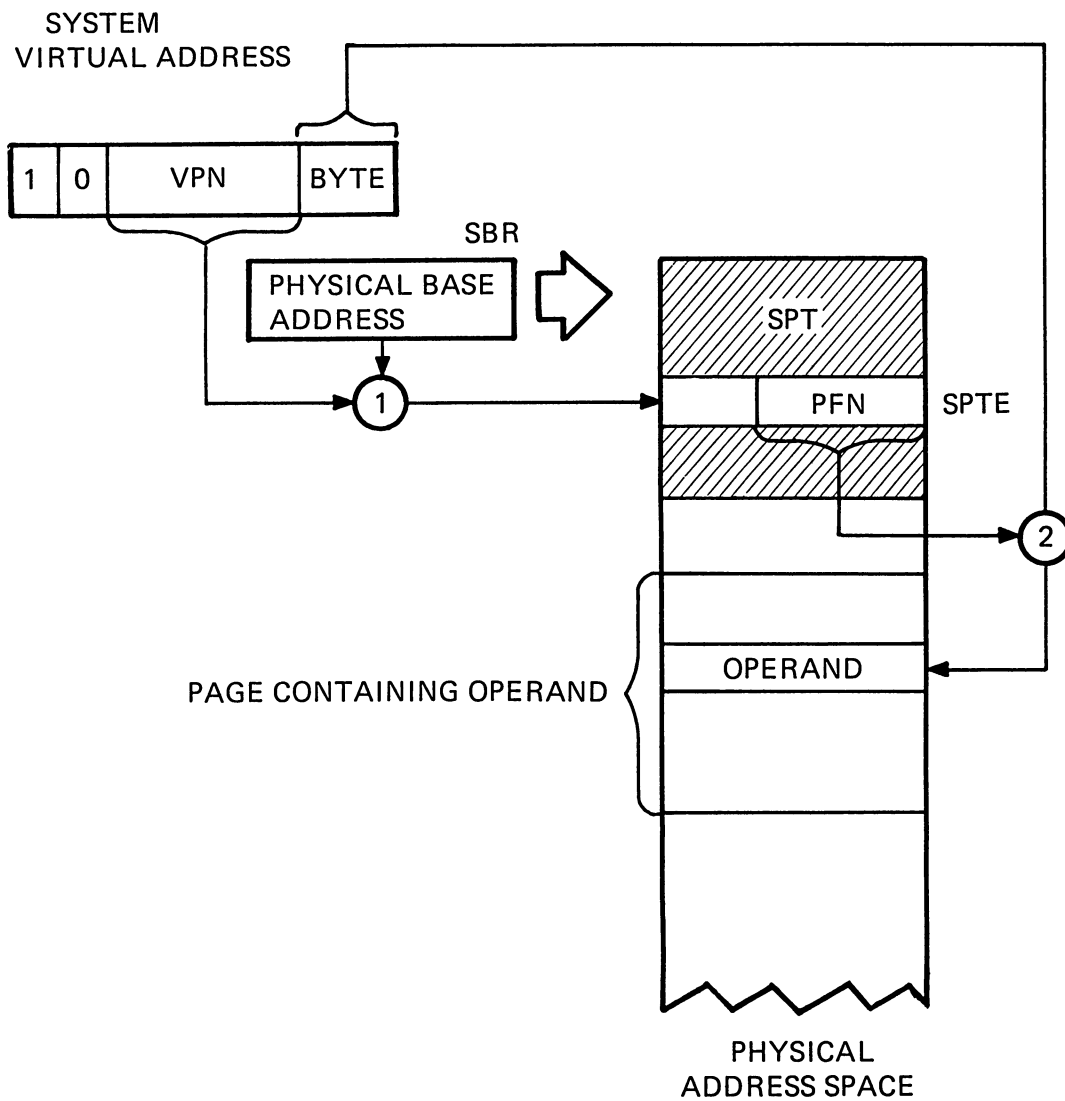
System Virtual Address Translation

The address translation mechanism recognizes a system virtual address when $VA\langle 31:30 \rangle = 10$. The virtual page number ($VA\langle 29:9 \rangle$) is used as an index into the system page table to locate the appropriate page table entry. The system page table can be found since the system base register contains the physical address of the base of the system page table.



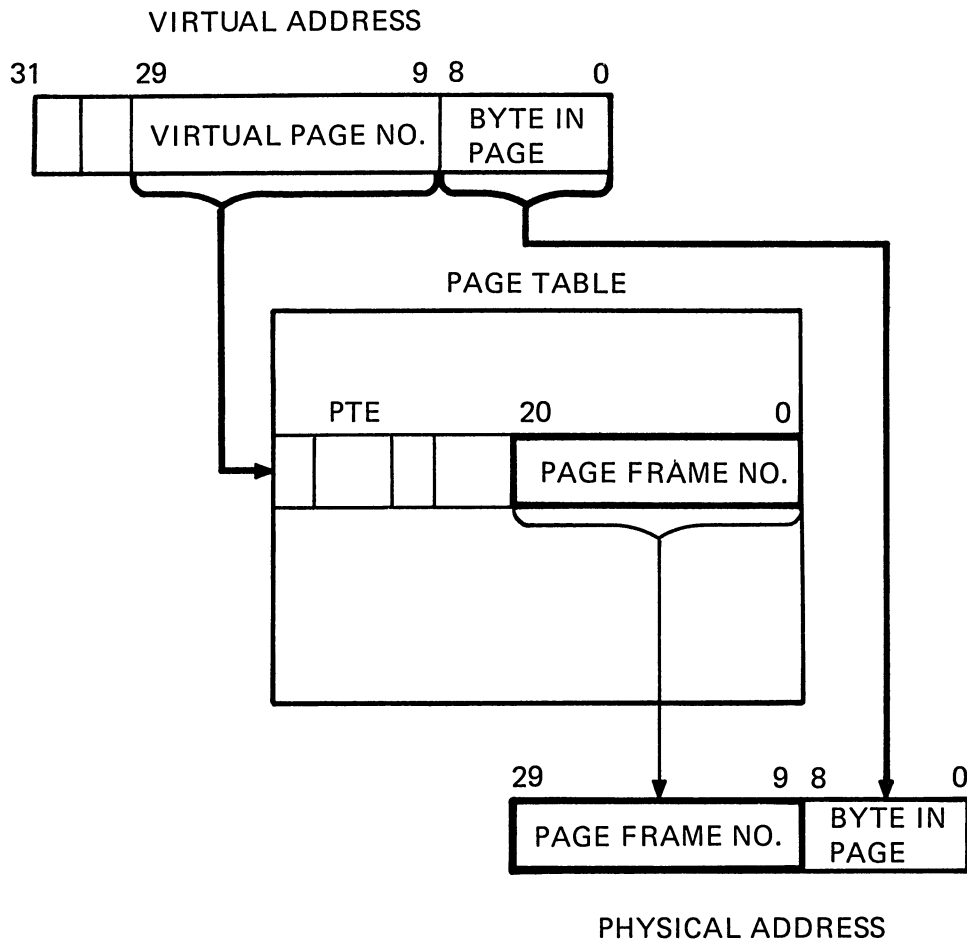
TK-3553

Figure 2-18. System Virtual Address



TK-3564

Figure 2-19. System Virtual Address Translation. Assuming that the valid bit is set, the PFN field can then be used to form the upper 21 bits of the physical address.

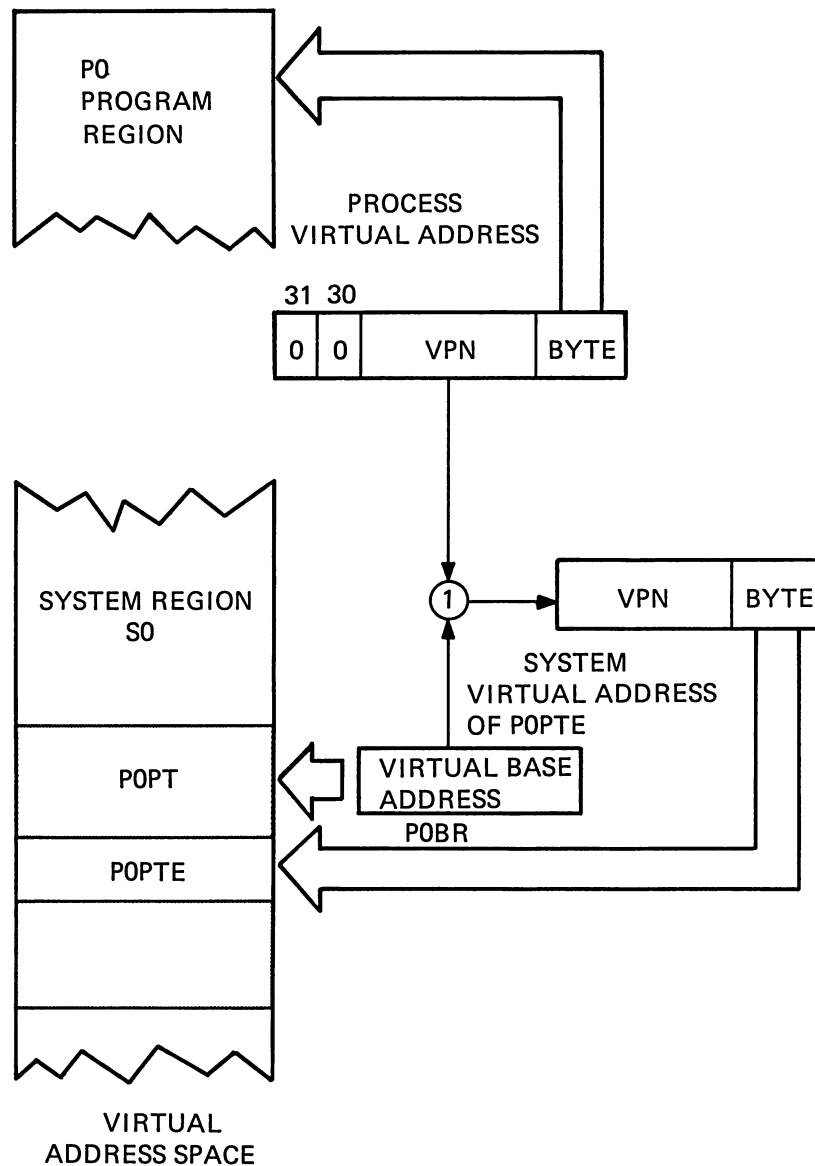


TK-3576

Figure 2-10. (Repeated)

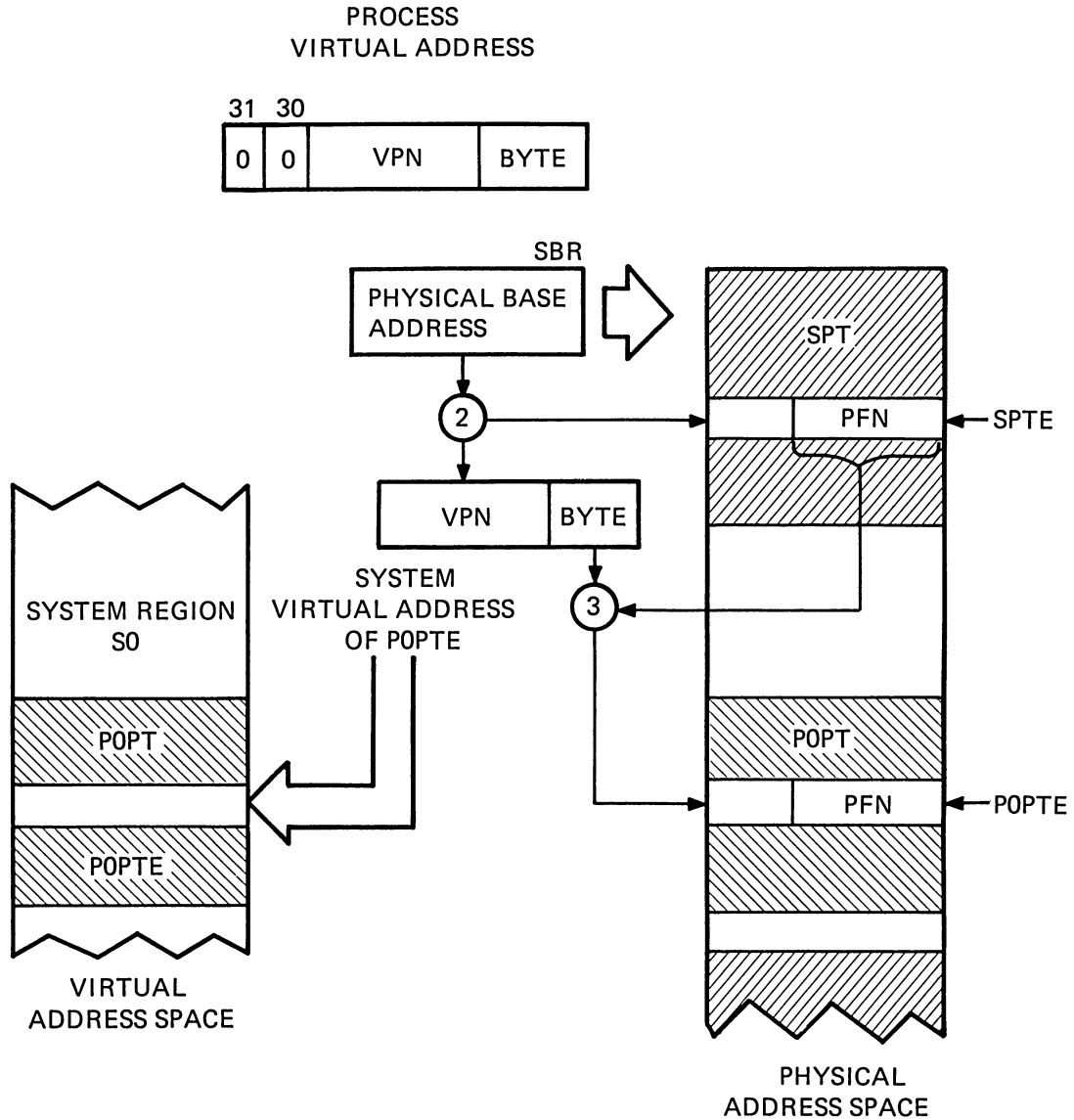
Process Virtual Address Translation

The translation of a virtual address in P0 space ($VA\langle 31:30 \rangle = 00$) or in P1 space ($VA\langle 31:30 \rangle = 01$) introduces added complexity. As before, the virtual page number is used as an index into the P0 page table or the P1 page table. However, since P0BR and P1BR contain virtual addresses, the resulting location of the P0PTE or P1PTE is its system virtual address, which must itself be translated. The following illustrates this with a pictorial translation of a P0 virtual address.



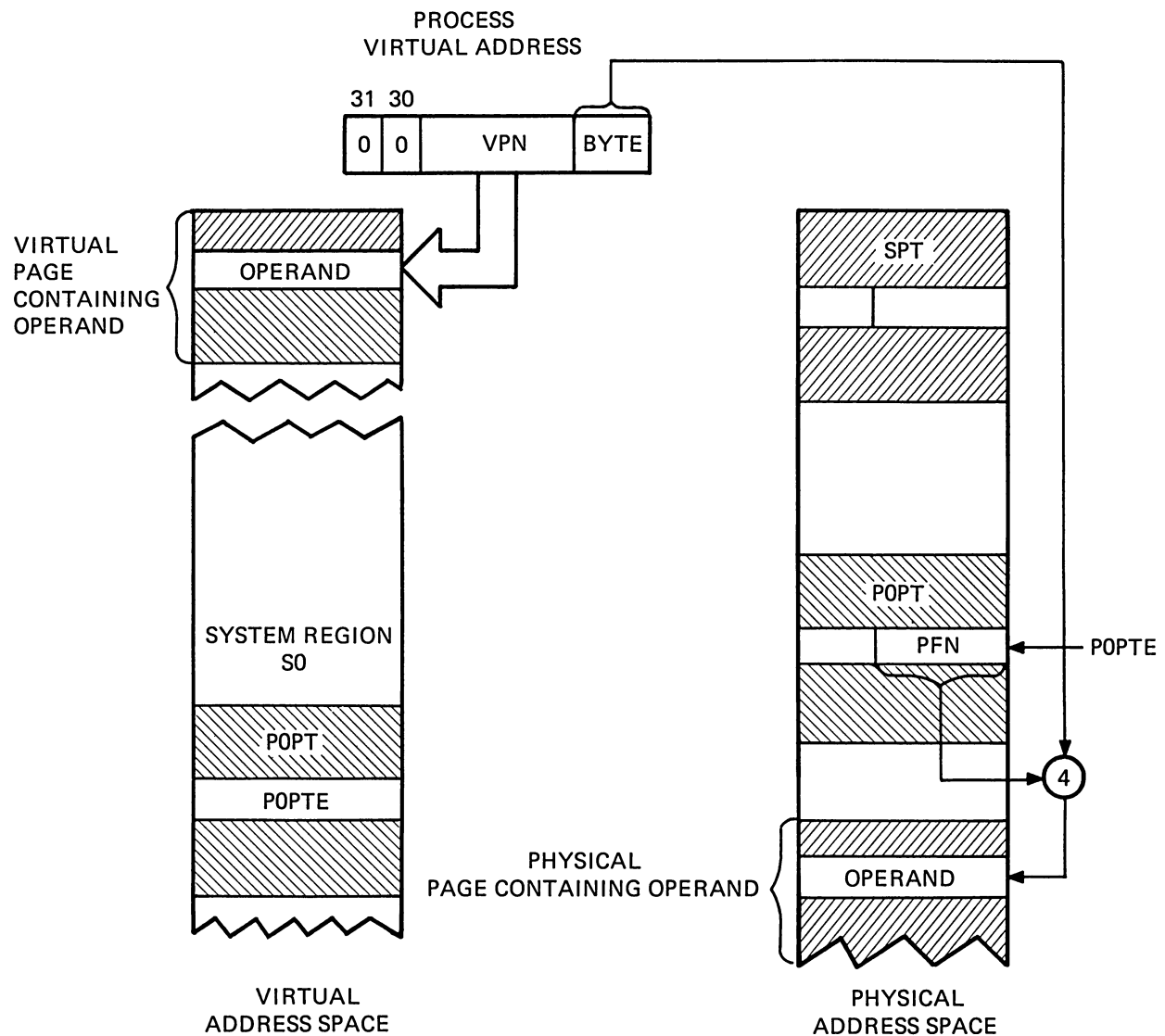
TK-3554

Figure 2-20. Process Virtual Address Translation (1). The virtual page number and the P0BR contents allow the appropriate P0PTE to be located in system virtual address space.



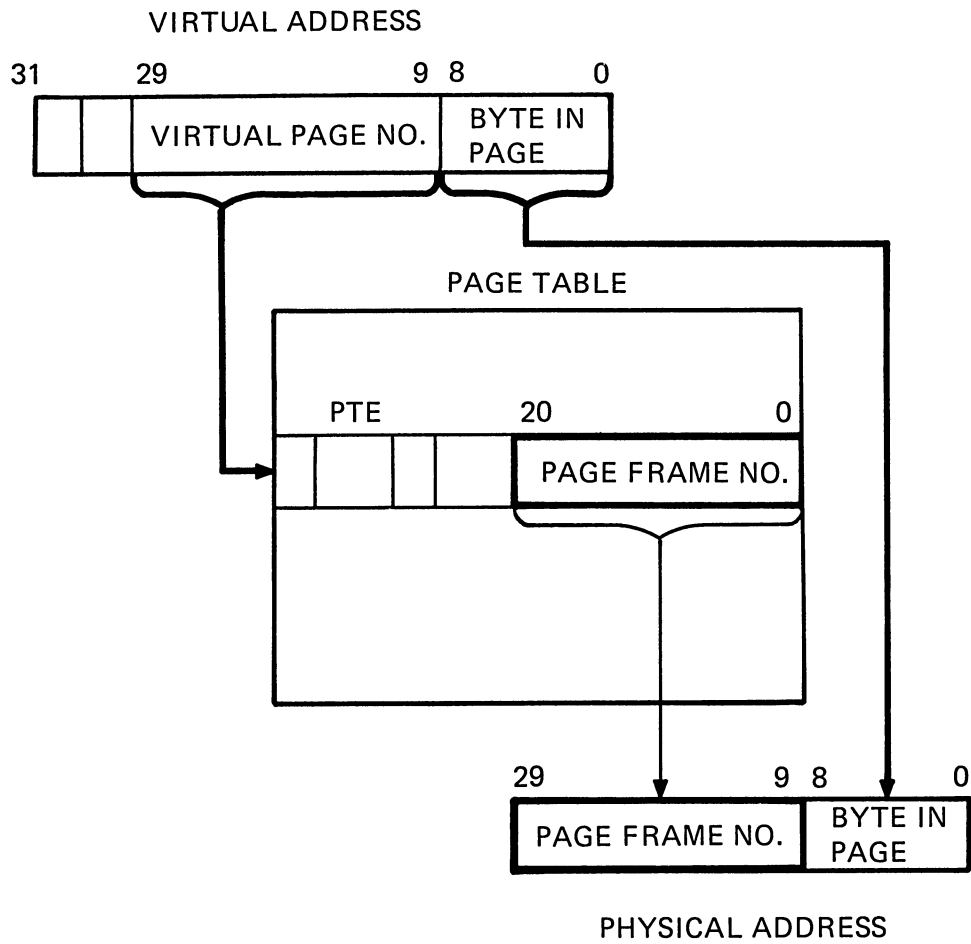
TK-3556

Figure 2-21. Process Virtual Address Translation (2). Before address translation can proceed, this page table entry must be located physically which requires the translation of its system virtual address. This operation is identical to the one described in the previous section.



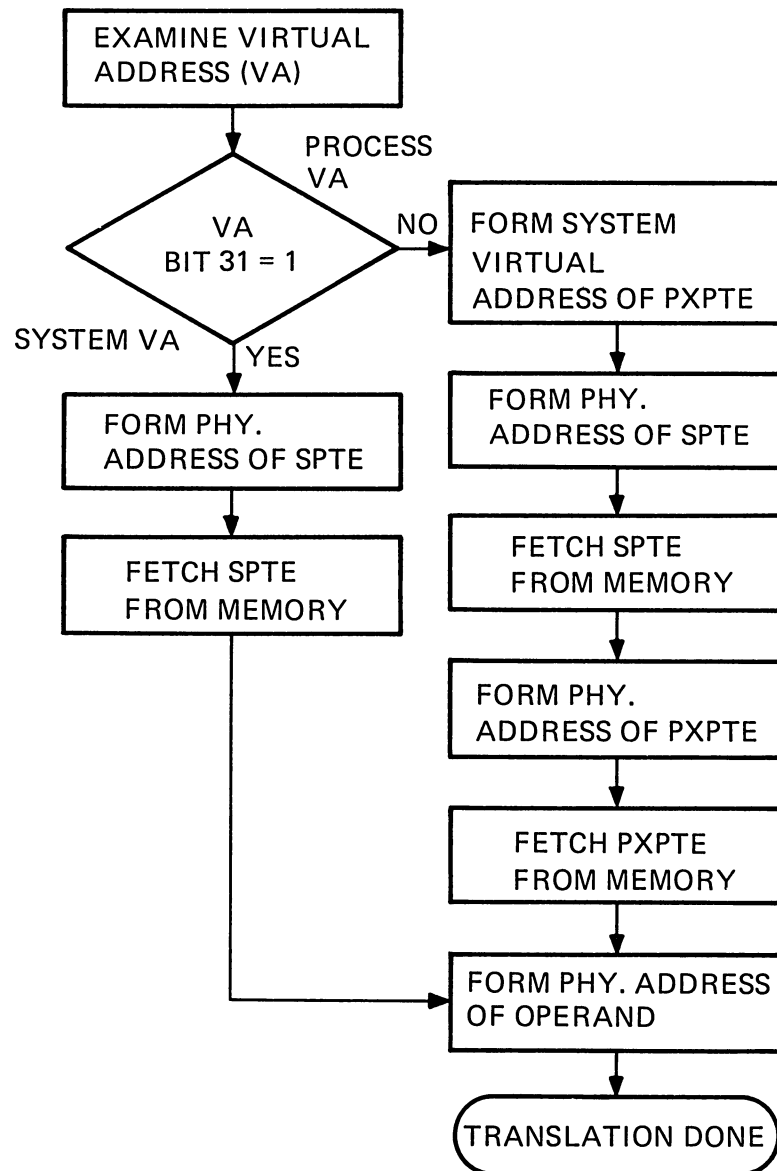
TK-3565

Figure 2-22. Process Virtual Address Translation (3). Once the **POPTE** has been located physically, the **PFN** field can be used (assuming that the valid bit is set) with the byte offset field from the process virtual address to form the physical address of the original operand.



TK-3576

Figure 2-10. Virtual Page Number Field (Repeated)



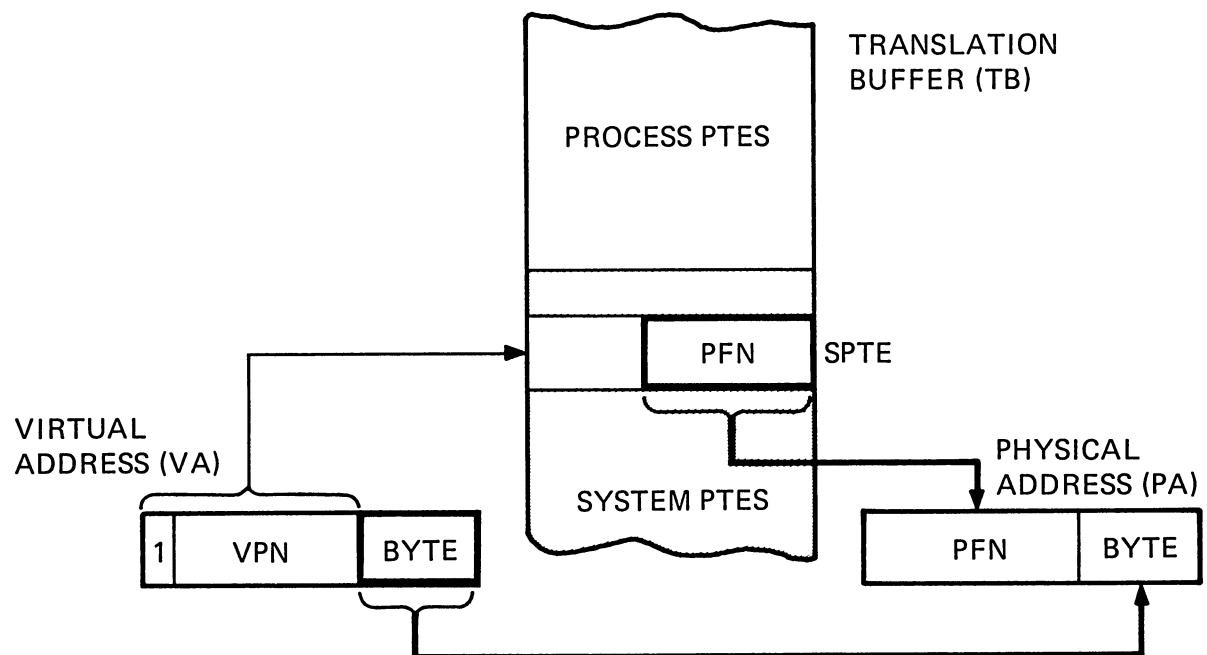
TK-3561

Figure 2-23. Process Virtual Address Translation Flow Diagram. Steps that may be necessary to translate virtual addresses into physical addresses. Note that a full translation of a process virtual address requires a system virtual address (of PxPTE) to be translated. For this reason, there are the same two steps along either path.

Translation Buffer

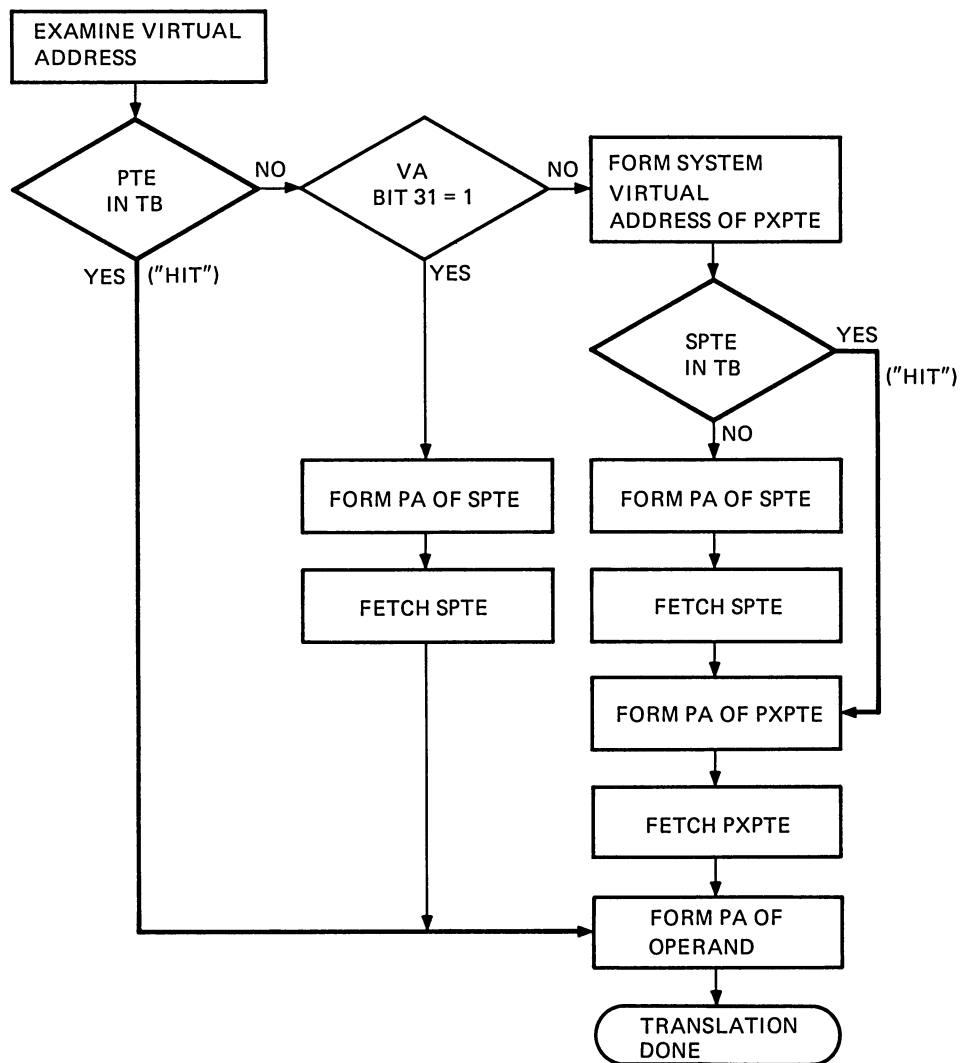
The number of steps required in address translation, especially of process virtual addresses, suggests the need for some means to speed up address translation. The translation buffer, a cache of page table entries, satisfies this need.

Once a virtual address is presented to the address translation mechanism, the page table entry required to translate is uniquely identified. A check is made to see whether the needed PTE is already in the translation buffer. (How this check is made will be discussed in the appendix to this module.) If so, a translation buffer hit, the physical address of the operand, can be immediately formed.



TK-3566

Figure 2-24. Translation Buffer "Hit" (PTE in TB)



TK-3563

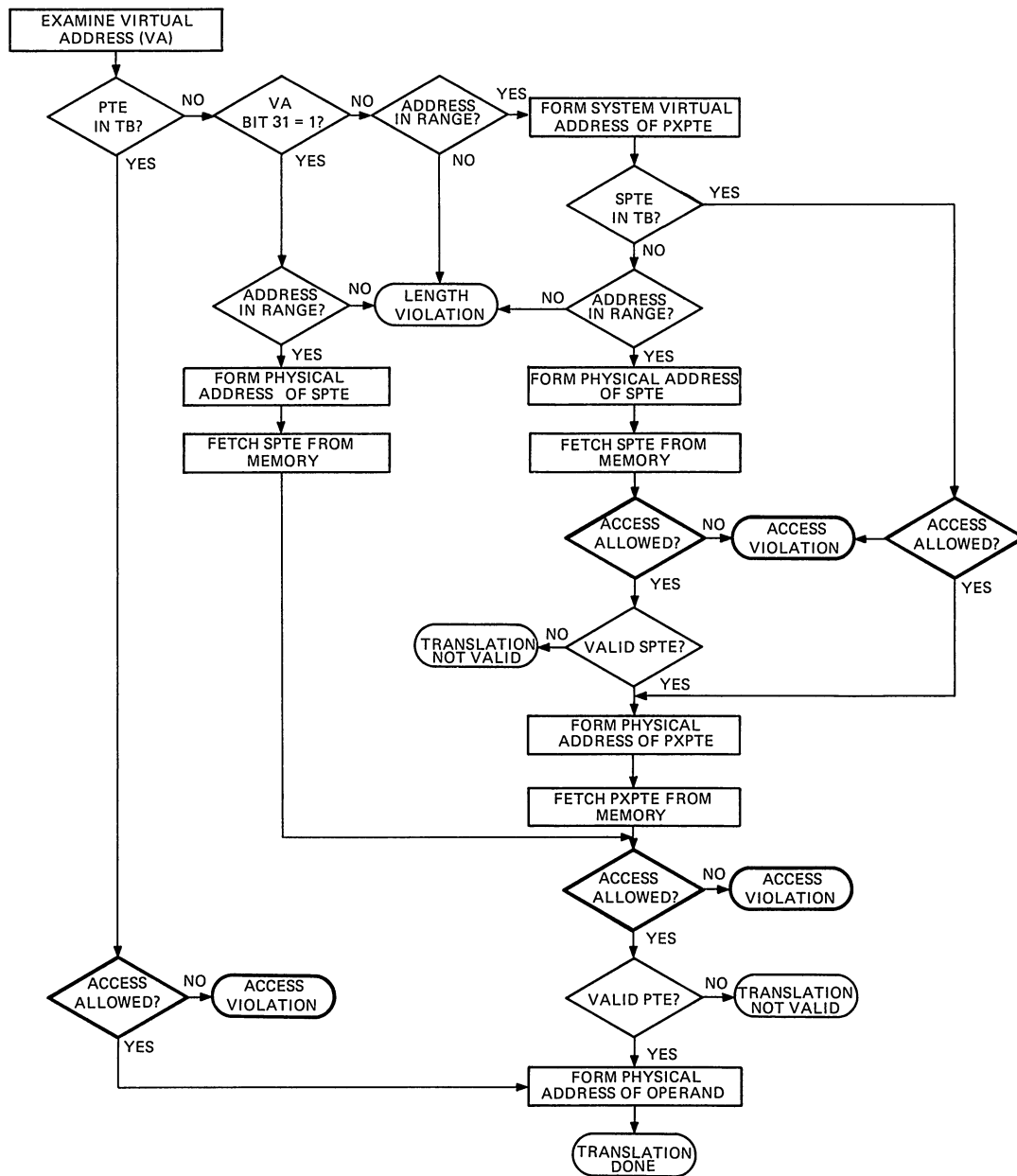
Figure 2-25. Translation Buffer Flow Diagram. If not a translation buffer miss, the page table entry must be retrieved from main memory and address translation follows the steps previously pictured. Note that the translation of a process virtual address actually requires two page table entries (PxPTE and the SPTE necessary to translate PxPTE). Thus a translation buffer miss on PxPTE could still result in a translation buffer hit on the SPTE.

Address Translation Faults

There are two possible faults that can occur while address translation is taking place.

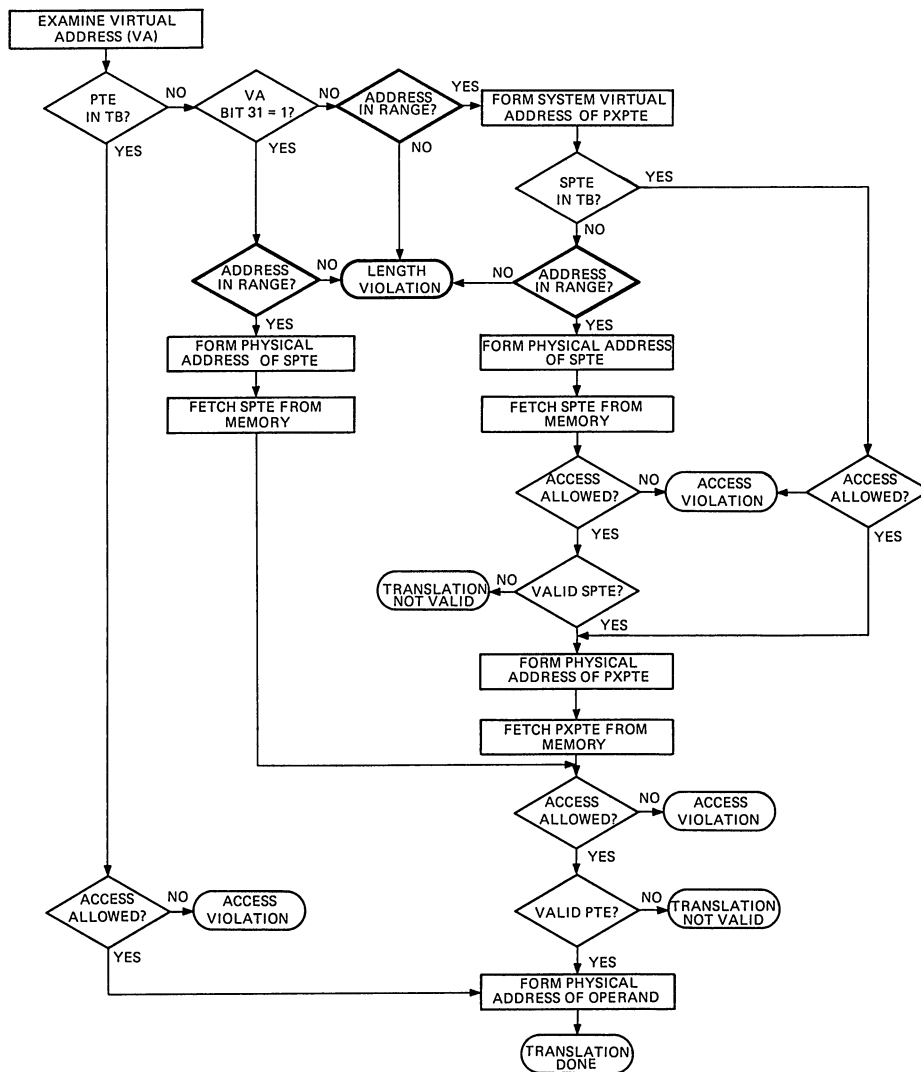
A protection code violation will occur if the requested access is not allowed for the current mode. The check is made against the protection field of the PTE, fetched from memory or retrieved from the translation buffer.

An additional protection check is made when translating process virtual addresses. The protection field of the SPTE which maps the PxPTE is checked to see if kernel mode can read the page, the minimum access above no access. By including this check, a page of process page table entries all specifying no access can be eliminated. (This check is missing in the corresponding figures in the workbook.)



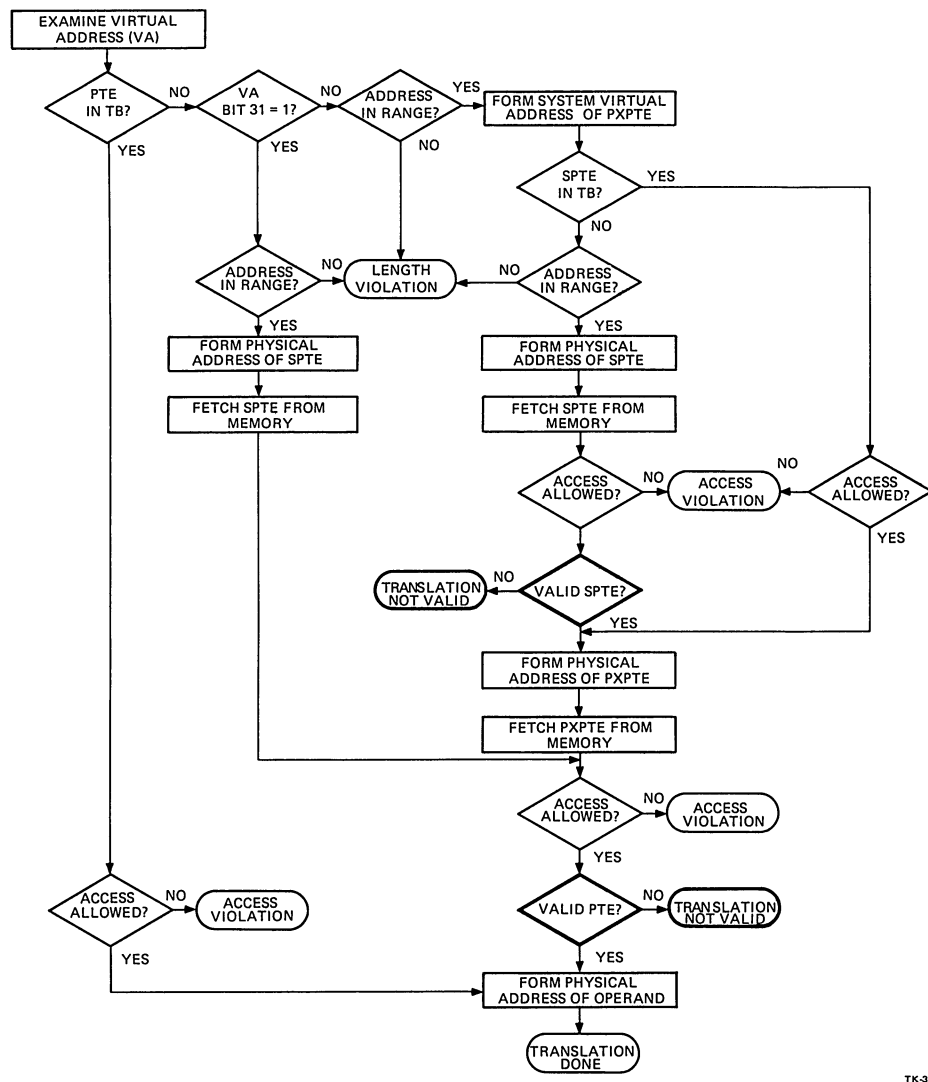
TK-3574

Figure 2-26. Address Translation Faults, Flow Diagram (1), Access Violation Fault



TK-3575

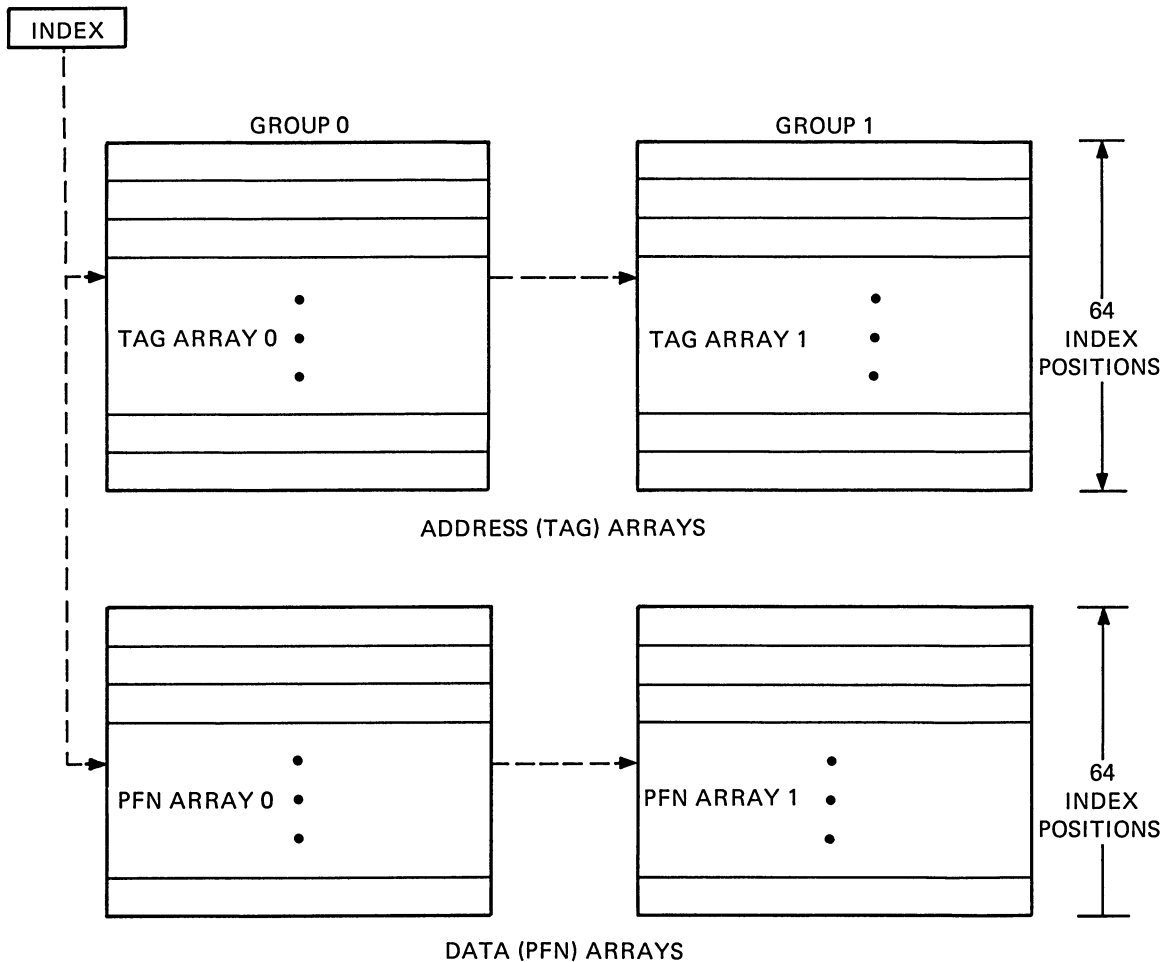
Figure 2-27. Address Translation Faults, Flow Diagram (2). Following a translation buffer miss, the address translation mechanism checks whether the page is in range by comparing the virtual page number to the appropriate length register. This check is unnecessary for the case of a translation buffer hit since the PTE would not be in the translation buffer if that page was out of range.



TK-3562

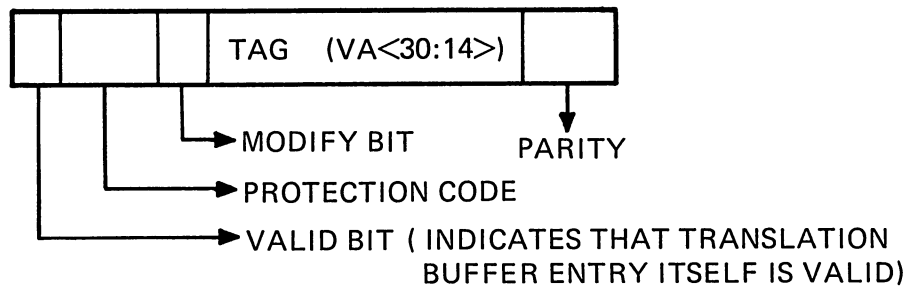
Figure 2-28. Address Translation Faults, Flow Diagram (3). Following a successful fetch of a PTE from memory and a protection code check, the setting of the valid bit is tested. If clear, a translation-not-valid fault occurs. As mentioned above, the validity check follows the protection code check.

APPENDIX A – TRANSLATION BUFFER

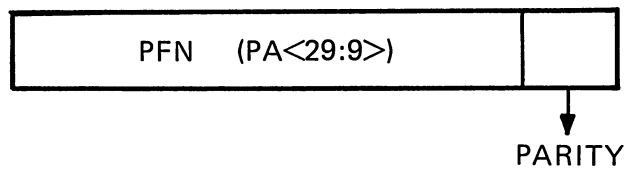


TK-3555

Figure 2-29. Translation Buffer. The translation buffer is a two-way set associative cache used to store page table entries and speed up address translation. The TB consists of two groups of two arrays: a data array containing the PFN field of the stored PTE and an address array containing a tag which identifies the PTE and the remaining fields of the PTE needed for address translation.



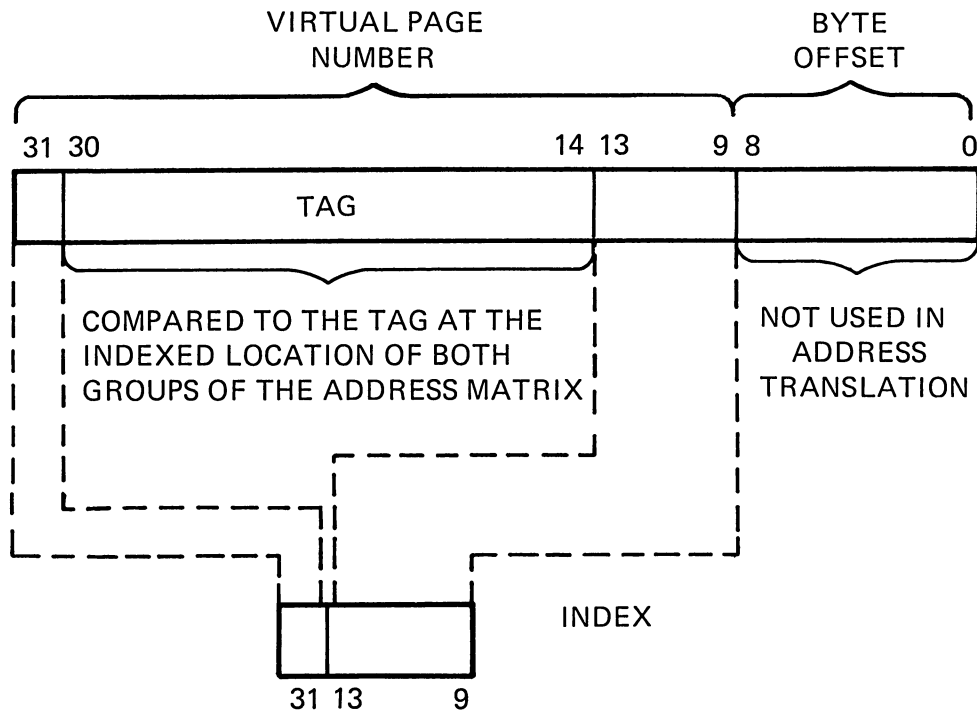
CONTENTS OF ADDRESS ARRAY ELEMENT



CONTENTS OF DATA ARRAY ELEMENT

TK-3557

Figure 2-30. Each group has an address array and a data array associated with it. The fields of a page table entry are found in pieces of both arrays.



TK-3558

Figure 2-31. The virtual address to be translated is broken up into three fields. The index field (VA<31>'VA<13:9>) will select an element in each of the four matrices. The tag field identifies whether the PTE stored in the TB is the one required for address translation. The index formed by VA<31>'VA<13:9> is used to select first an address matrix element and possibly a data matrix element from group 0 and then group 1.

When an address must be translated, the index field selects an element from the group 0 address matrix. If the tag found there matches the tag formed from the virtual address, access checks can proceed and the physical address formed from the corresponding element in the group 0 data matrix. If no match occurs in group 0, the tag field in the same element in the group 1 address matrix is checked. Again, if a match occurs, address translation can proceed.

A miss in group 1 is a translation buffer miss. The PTE must be fetched from main memory (which might also involve address translation). If the valid bit is set in the fetched PTE (valid bit clear results in a translation-not-valid fault), the PTE is stored in the correct location in either the group 0 matrix or the group 1 matrix. (The choice is random.) Once the PTE is in the TB, address translation is restarted with a guaranteed TB hit.

Note that a TB hit implies that the valid bit (of the PTE) is set. The valid bit in the TB governs whether that TB matrix element is valid. There are three cases when a TB matrix element must be invalidated.

1. At system startup time, the entire translation buffer must be cleared by writing to a special processor register

MTPR #1, #PR\$_TBIA

the translation-buffer-invalidate-all register.

2. Whenever software changes a PTE, that PTE must be invalidated if it is in the TB. If it is found in the TB

MTPR address, #PR\$_TBIS

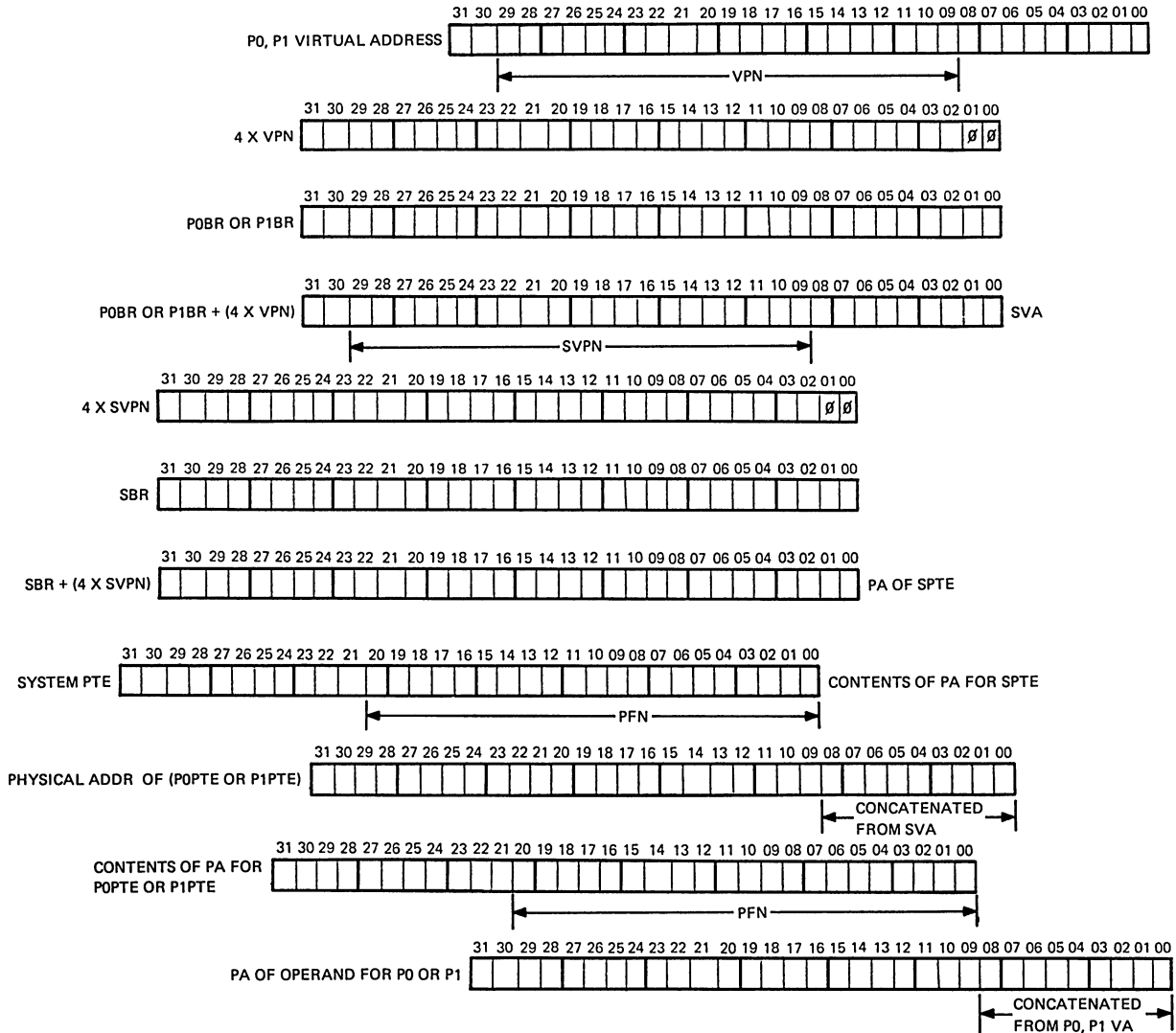
writes the address to the translation-buffer-invalidate-single register. If the associated PTE is in the TB, the entry is invalidated.

The most common example of this case is the removal of a page from the working set. The SETPRT system service will also invalidate a TB entry since this service changes the protection code in the PTE.

3. When an LDPCTX instruction is executed, the per-process half of the TB is invalidated. (By using bit 31 as one of the bits forming the index, the TB is effectively divided into a system half of 64 entries and a process half of 64 entries.)

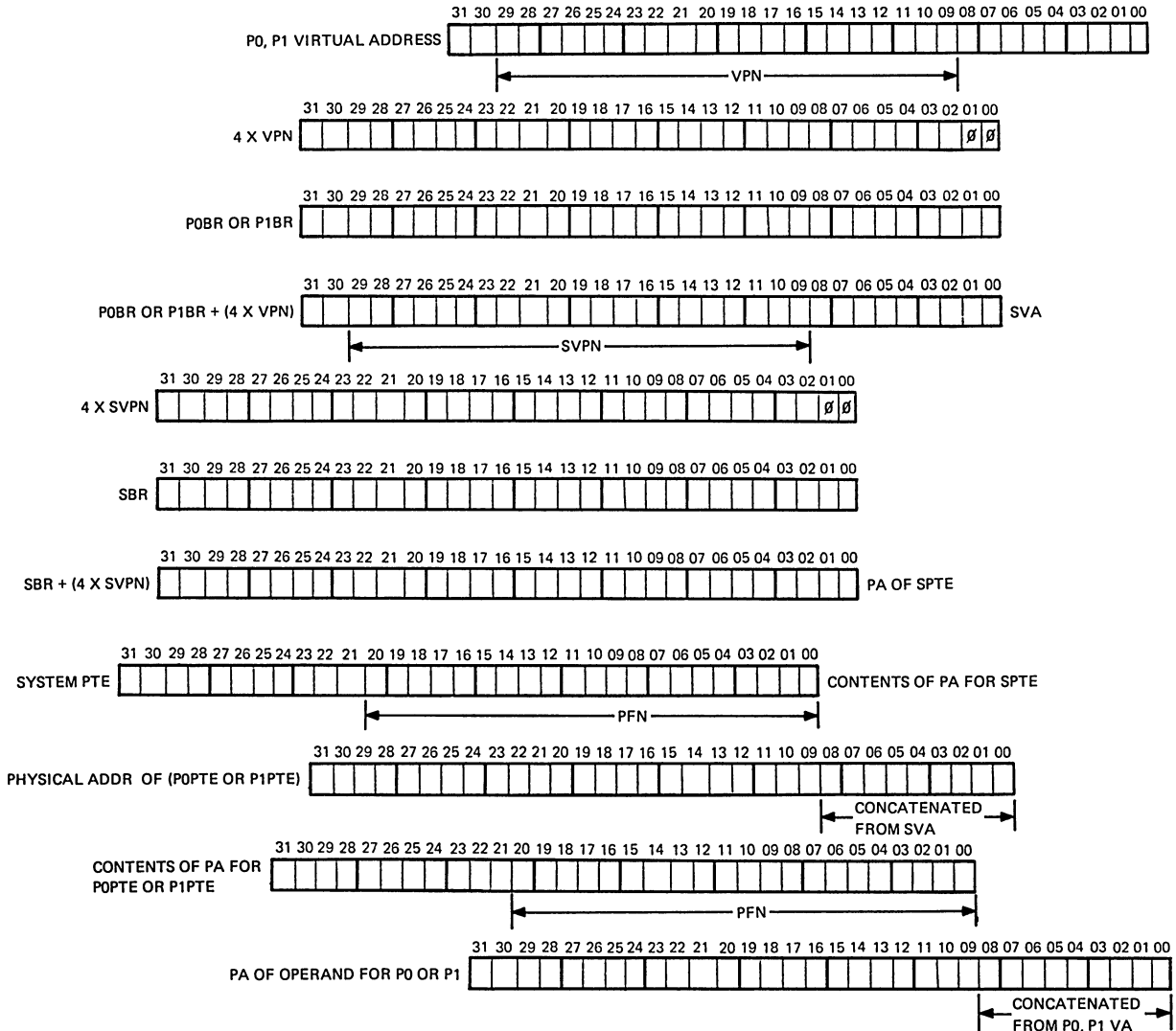
APPENDIX B – WORKSHEETS

MEMORY MANAGEMENT WORKSHEET FOR P0, P1



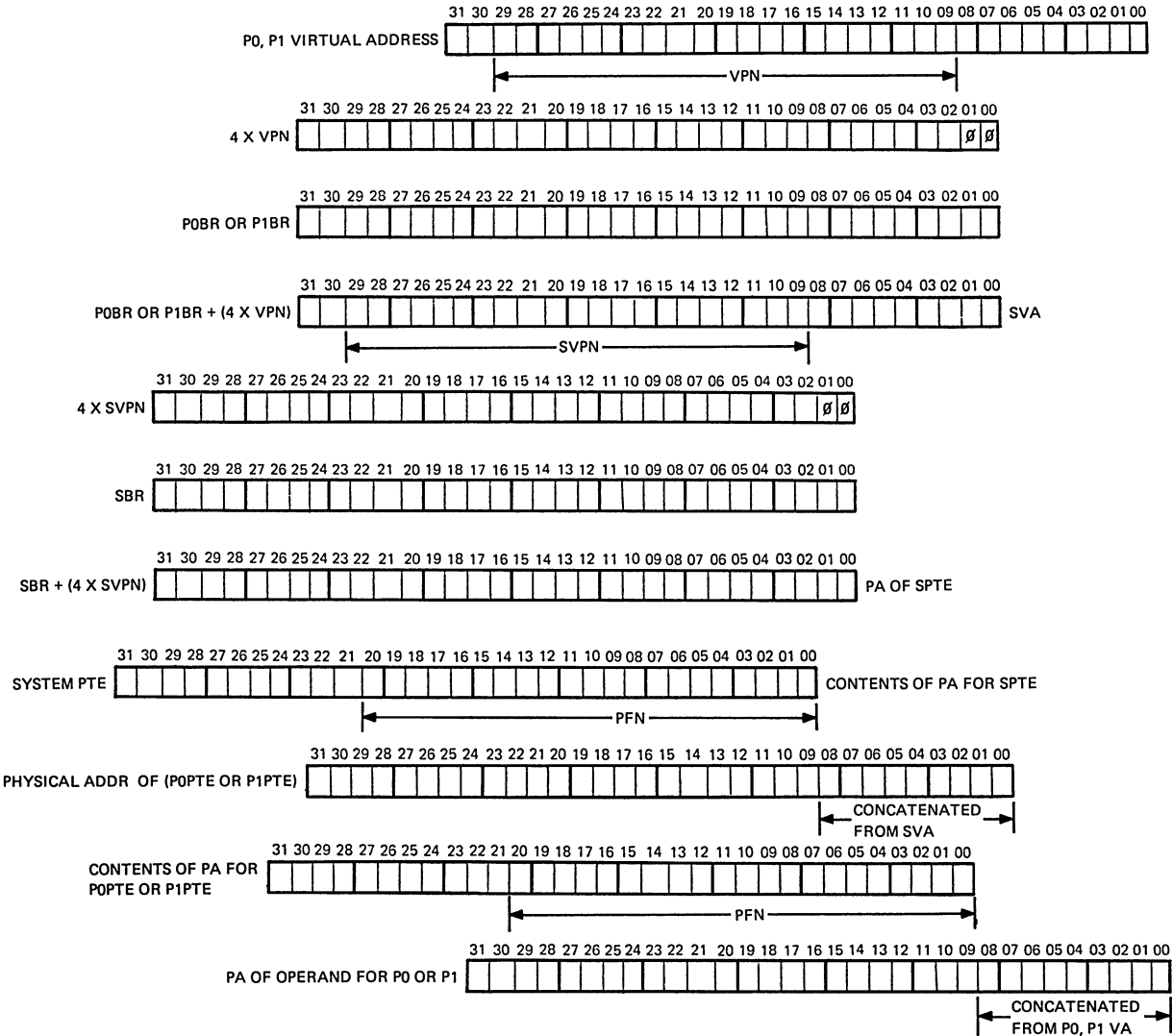
TK-1009

MEMORY MANAGEMENT WORKSHEET FOR P0, P1



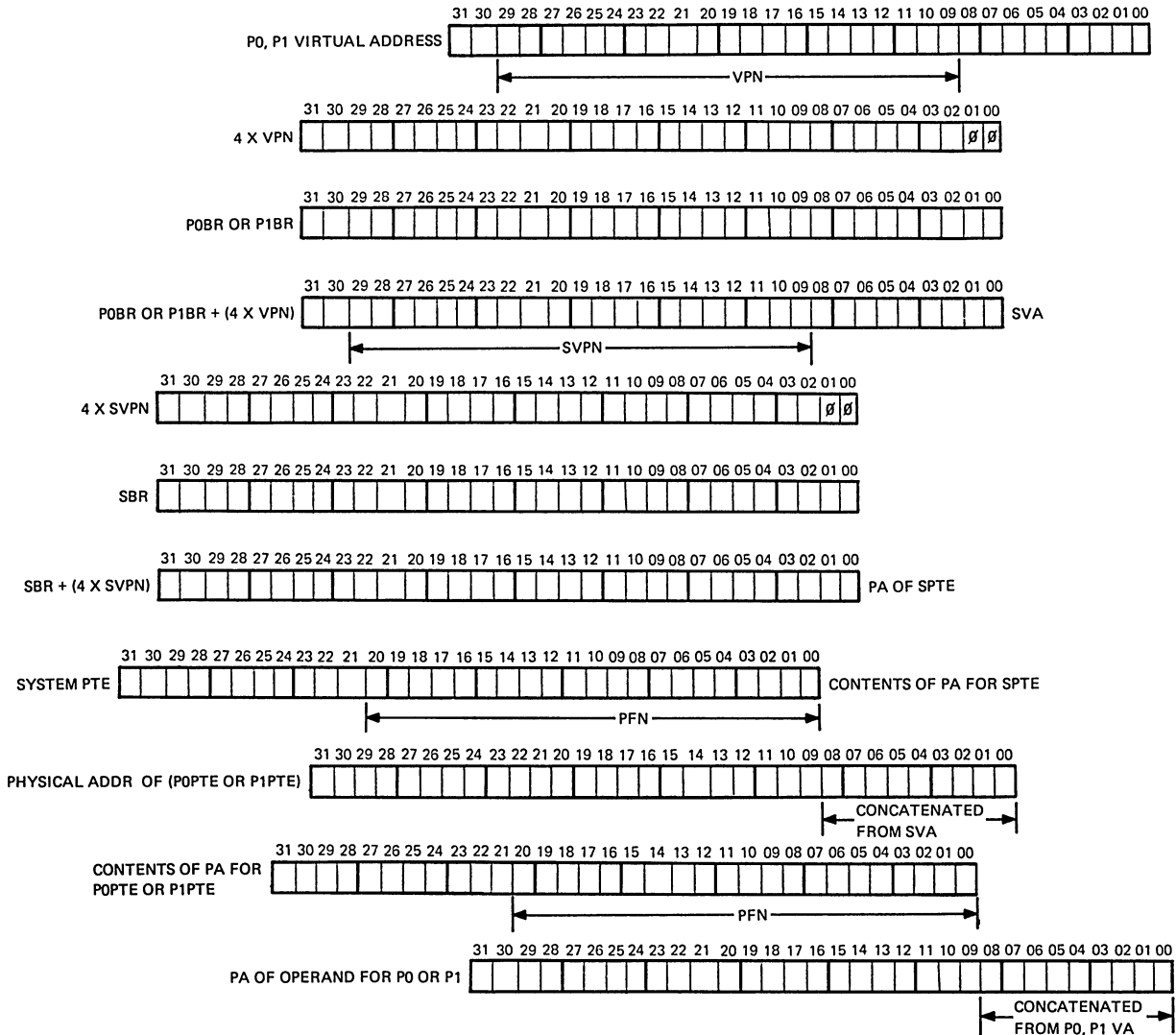
TK-1009

MEMORY MANAGEMENT WORKSHEET FOR P0, P1



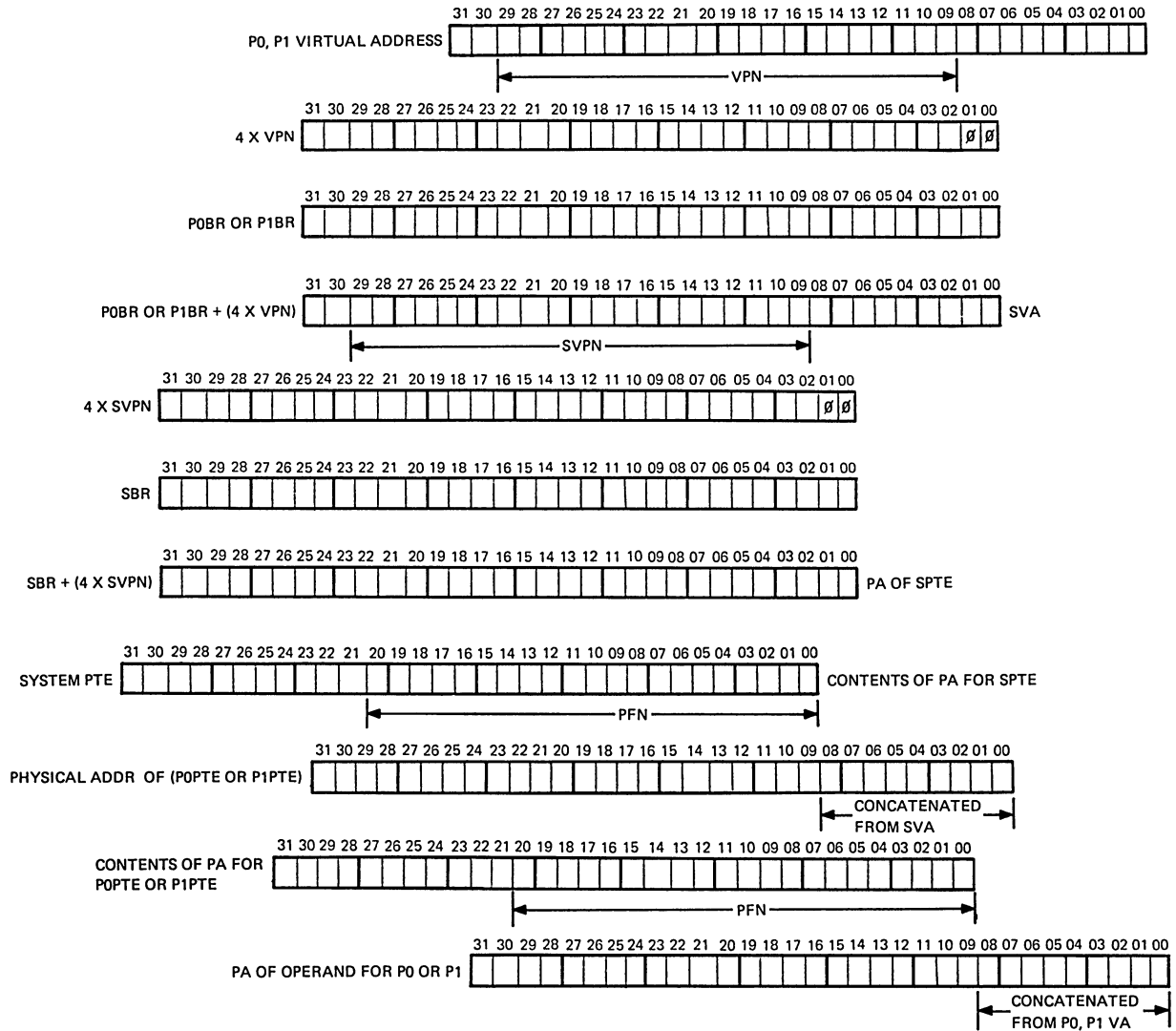
TK-1009

MEMORY MANAGEMENT WORKSHEET FOR P0, P1



TK-1009

MEMORY MANAGEMENT WORKSHEET FOR P0, P1



TK-1009

BLANK

MODULE TEST

The situation: You are trying to repair a down VAX-11/780 system with a partner who considers himself an expert in all subject matter relating to the 11/780 system.

There have been times when your partner has shown great genius in recalling specific facts relating to address translation that VMS needs to function. On the other hand, there are times when his facts were very misleading and cost you hours of repair time tracking down the wrong problem.

This test was written specifically for the time when you are under pressure to bring the system up and are forced to make decisions on the so-called expert statements of your partner.

The test questions have been written in two parts:

Part one is the concept or statement that you know is 100 percent correct.

Part two is the statement that your partner makes to you relating to the fact that you know.

You must judge that statement to be true or false.

1. Virtual address space can be divided into two pieces by using bit 31 of the virtual address.

Statement: When bit 31 is set, the virtual address is a process virtual address found in process space.

TRUE FALSE

2. Address translation involves transforming a 32-bit virtual address into a 30-bit physical address.

Statement: By setting the page table entry appropriately, any virtual address can point to either main memory or I/O addresses.

TRUE FALSE

MODULE TEST

3. The operating system keeps track of the status and physical location of virtual pages in a set of data structures called page tables.

Statement: Protection information is also found in these structures.

TRUE FALSE

4. The system page table is built at initialization time and is located in contiguous pages of physical memory. Process page tables are set up at process creation and altered at image activation, image exit, and in response to various system services.

Statement: Process page tables are located in virtually discontiguous pages of system space.

TRUE FALSE

5. The most significant bit in the PTE is called the valid bit. When this bit is set, the appropriate virtual page is in the working set. In addition, when a page is valid, the M-bit and the PFN field may be used by memory management hardware. The modify bit indicates whether the page has been modified since last brought into the working set.

Statement: The PFN field indicates which virtual page is mapped to by this virtual address.

TRUE FALSE

6. The information contained in the page tables is used not only by the operating system but also by the address translation hardware. To provide a simple means to locate the page tables, the processor contains three registers containing the base addresses of the three page tables.

Statement: All base registers must contain physical addresses.

TRUE FALSE

7. During address translation, two different kinds of exception can occur: access violation and translation-not-valid exception.

Statement: Both forms of these exceptions are classified as faults.

TRUE FALSE

MODULE TEST

8. During address translation, two different kinds of exception can occur.

Statement: The address translation mechanism checks the protection code before it checks the valid bit.

TRUE FALSE

9. List two conditions that would create a memory management exception.

10. Given the following data:

a.	P0BR/80005000	P0LR/A
b.	P1BR/7F808004	P1BR/1FFFF0
c.	SBR/1400	SLR/F
d.	PA 1400/A0000004	
e.	PA 14A0/A0000005	
f.	PA 1500/A0000007	
g.	PA E00/A0000003	
h.	PA A00/A0000001	
i.	PA A04/A0000002	

virtual address 250 will translate to what physical address?

BLANK

CONSOLE INTERFACE BOARD

INTRODUCTION

The console interface board (CIB) links the console subsystem to the VAX-11/780 central processor.

This lesson will provide the student with an understanding of the CIB bus structure, hardware registers, and the proper communication protocol needed between the console subsystem and CIB.

3

OBJECTIVES

1. Given a CIB function, identify the registers needed to implement it.
2. Develop and implement a program under octal debugging technique (ODT), that will execute a console function.
3. Identify the function of the following buses:
 - a. Q-Bus
 - b. ID-Bus
 - c. V-Bus

SAMPLE TEST ITEMS

There will be a performance test to evaluate your understanding of the objectives. The instructor will evaluate the printout that you will obtain during this test.

1. Using ODT, write a program in machine (PDP-11) language to enable the user to stop the VAX PCS in ROM state 01F8. ROM state 01F8 is a state within the initialize sequence of the VAX-11/780 PCS which starts at 100.

This program must start the sequence at 100 and stop on a micromatch at 1F8. Start your program at LSI memory location 1000.

RESOURCES

1. KC780 Console Interface Board Technical Description
2. KA780 Engineering Drawings (M8236 - CIB)

LECTURE OUTLINE

- I. Console Interface Board Function
- II. Functions Implemented by the Hardware
 - A. Clock functions
 - B. Reading/writing ID registers
 - C. V-bus registers
- III. Console Functions Implemented by Microroutines
 - A. Examine/deposits
 - B. Continue
 - C. Quad-clear
 - D. SBI unjam
- IV. Communication Protocol Interaction Between the LSI-11 and VAX-11/780
 - A. Simulate console command >>>D/ID 21 62
 - B. Simulate console command >>>E/ID 21
 - C. Simulate console command >>>E R5
- V. Laboratory Session

BLANK

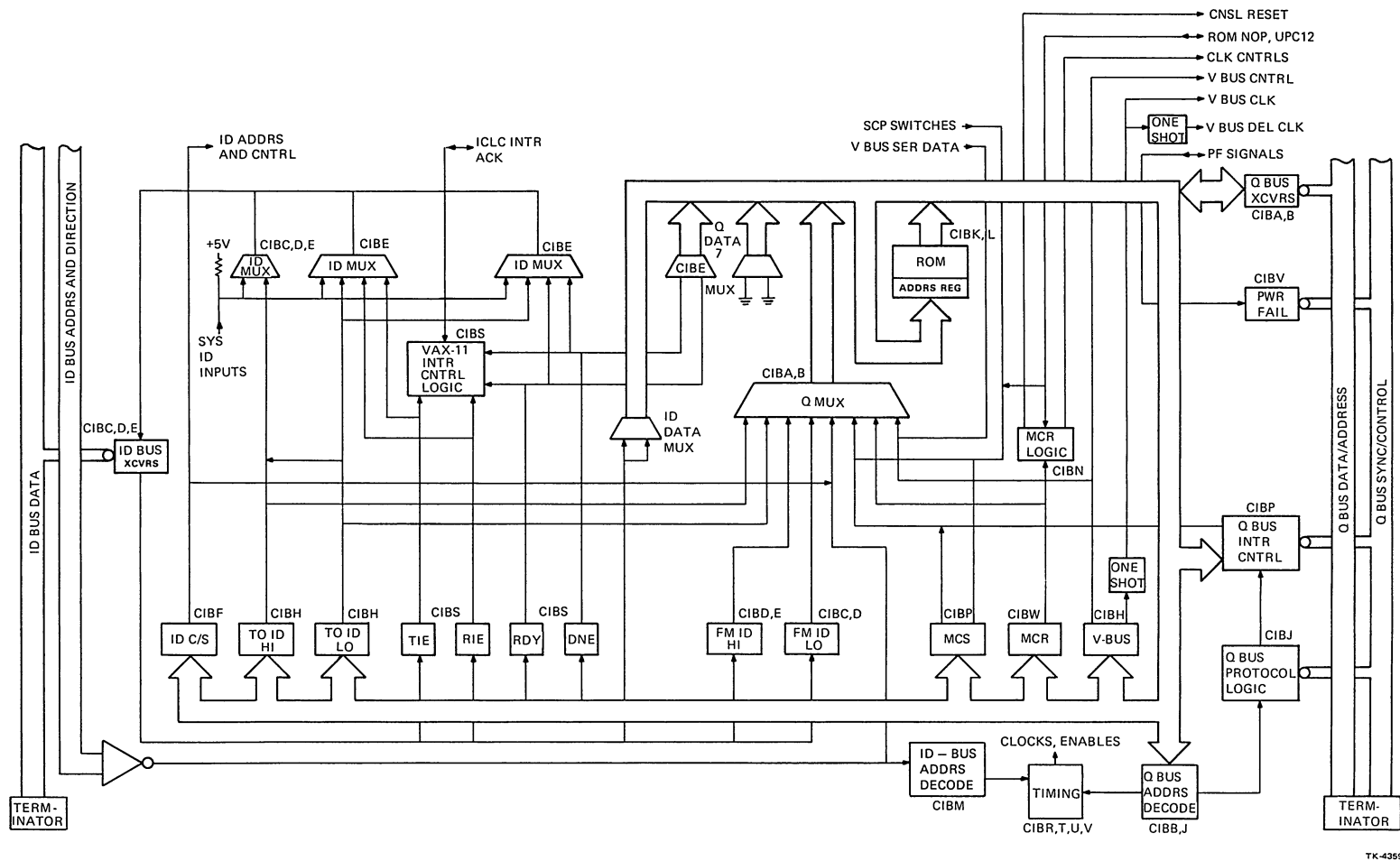
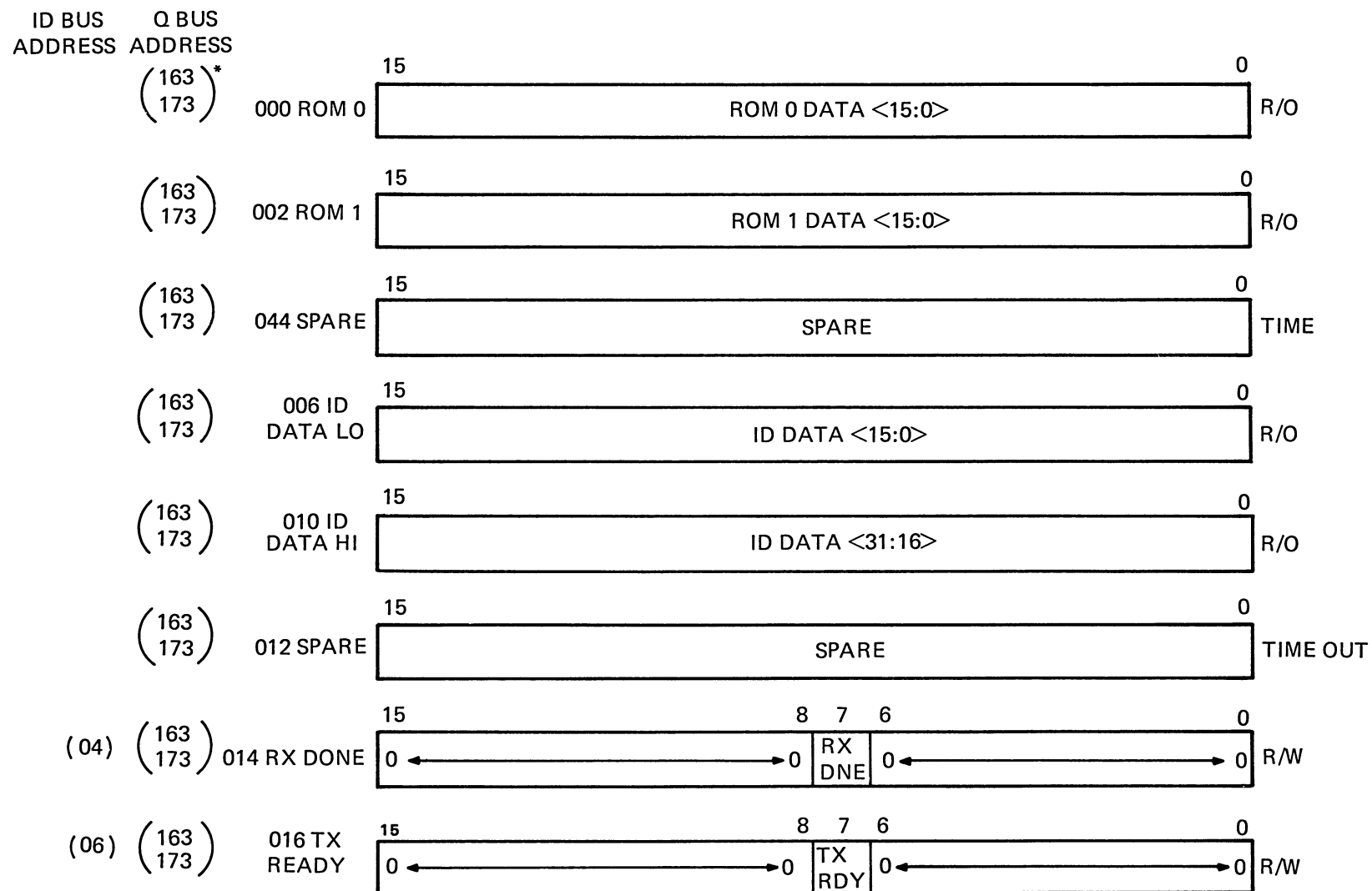


Figure 3-1. Console Panel Interface

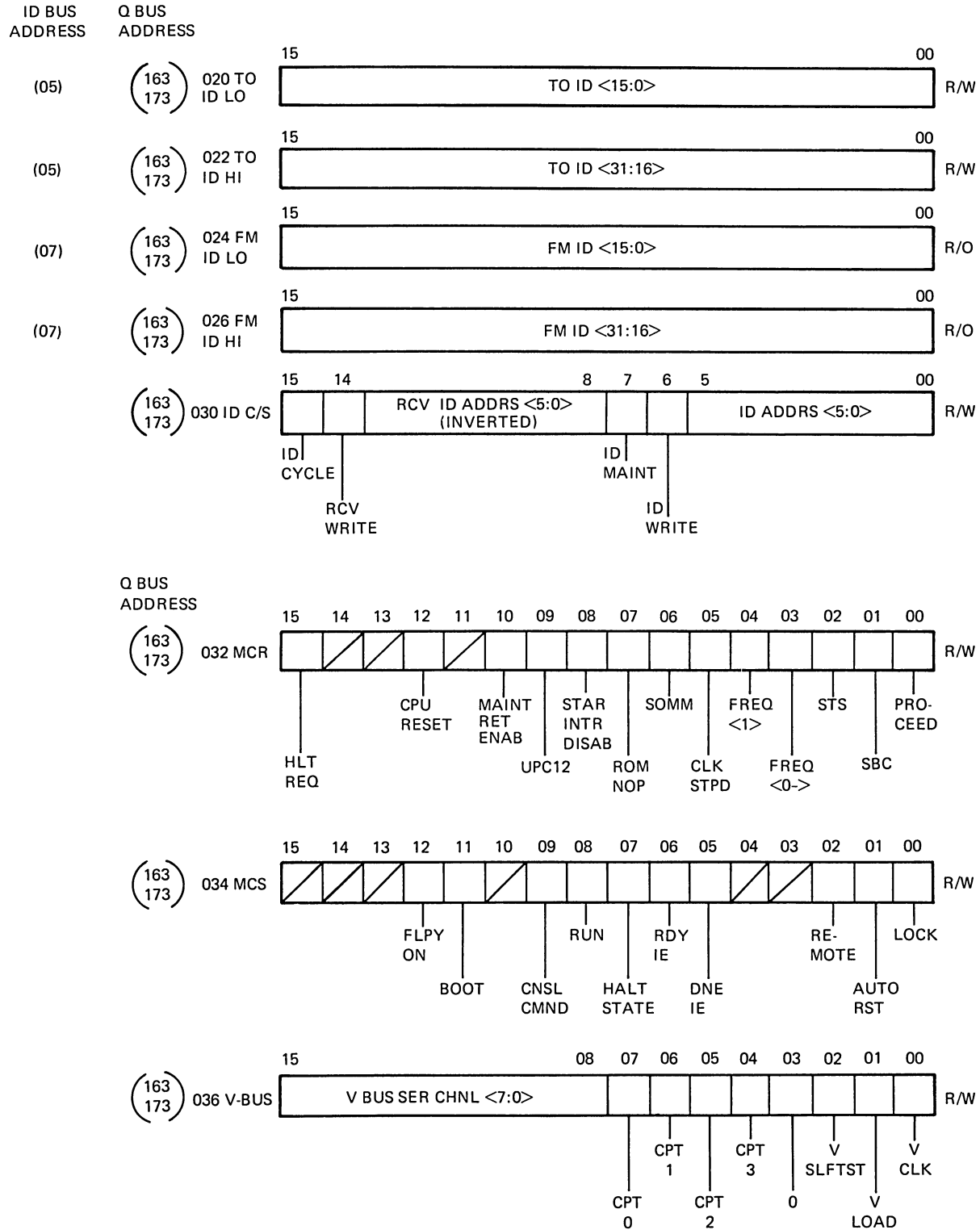


*START ADRS DETERMINED BY
JUMPER W1 ON M8236. SEE PAGE
CIBB OF M8236 PRINTS FOR
JUMPER DEFINITION'

TK-0204

Figure 3-2. Q-Bus Registers, Lower Eight

Console Interface Board



TK-4356

Figure 3-3. Q-Bus Registers, Upper Eight

BLANK

LAB EXERCISES

The objective of this lab is to enable the user to perform various console functions using only the ODT language of the LSI.

The exercises for this lab are designed such that each skill developed in one lab will be needed for the next lab. All examples will be covered in a classroom environment. The lab sessions are for the student to explore ODT and its possibility as a troubleshooting aid. Do not insert a floppy diskette for ODT exercise.

Reference materials

1. CIB Technical Manual
2. KA 780 engineering prints
3. LSI Handbook

EXERCISE 1

1. Write a sequence that will deposit 62 (base 16) to ID register 3C.
2. Write a sequence that will enable you to examine ID register 3C.
3. Write a sequence that will clear out ID register 3C.
4. Write a sequence that will deposit 80005000 to P0BR on the ID Bus.
5. Write a sequence that will allow you to examine the P0BR.
6. Write a sequence that will clear out the P0BR.

These exercises (which are not complex) are needed to develop the basic understanding for other labs. Before doing exercise 2, try a few more deposits, examines, and clears to other ID register to fortify your own understanding.

EXERCISE 2

1. Write a sequence that will put VAX CPU into single time state.
2. Write a sequence that will allow you to observe movement of a bit through CPT time states. (Hint: This should be done with a deposit, then examine, then deposit, etc., of the correct register.)

3. Write a sequence that will clear out those registers when done with the above exercise.
4. Read the description for single bus cycle, then write a sequence to perform SBC. What can you do to prove that SBC happened?
5. How would you clear clock stop bit <5> of MCR (173032)?
6. What is one way of ensuring that VAX CPU clock will always stop in CPT0?

EXERCISE 3

1. Write a sequence that will enable you to examine an ID register using single time-state. Write bit <31> of ID register 1D. The registers that must be examined to prove this process are:
 - a. 173026
 - b. 173036
 - c. 173030
 - d. 173032

EXERCISE 4

1. Here is a program in machine (PDP-11) language to enable the user to stop the VAX PCS in a known ROM state. The ROM state selected is in the initialize routine (microword 985). Code and comment the following program:

```

1000      MOV #4605, (R0)      /
1004      MOV #100141, (R1)   /
1010      MOV #400, (R0)      /
1014      MOV #100140, (R1)   /
1020      MOV #2101, (R2)     /
1024      HALT

```

Contents of GPR R0-R2

R0/173020 R1/173030 R2/173032

Deposit this program in the LSI using ODT. Start the program and verify that the program works by checking the lights of the micro-PC (Slot 23).

2. Develop a program in PDP-11 language to deposit and examine an ID register. Deposit and run your program for verification. You can move the contents to a GPR.

EXERCISE 5

In this exercise you will simulate console commands using your knowledge of ODT and PCS listings. Refer to ESKAA 4.3, 1 of 2 I5 for data type.

Simulate the following:

1. >>>D AA 500
2. >>>E AA
3. Deposit, using console command language, the following program in VAX memory. DO NOT EXECUTE.

```
1000/ MOVL #1000, SP
      MOVL #12345678, R2
      HALT
```

Also set up your trap catcher using CCL by >>>D 0 3/N:80 and then >>>D/ID 3B 0.

Generate a PDP-11 program (LSI) to simulate the following:

- a. >>>D PC 1000
- b. >>>C
- c. >>>E R2

The program in VAX memory should be executed and 12345678 will be in VAX GPR R2.

This exercise may, in the future, serve as a springboard for you to develop other routines to aid your troubleshooting techniques.

BLANK

MODULE TEST

This is a performance test to evaluate the module objectives. The instructor will evaluate the printout that you obtain during this test.

1. Using ODT, write a program in machine (PDP-11) language to enable the user to stop the VAX PCS in ROM state 01F8. ROM state 01F8 is a state within the initialize sequence of the VAX-11/780 PCS which starts at 100. This program must start the sequence at 100 and stop on a micromatch at 1F8. Start your program at LSI memory location 1000.
2. Using ODT, write a program in machine (PDP-11) language to simulate this series of console commands.

```
>>>I  
>>>U  
>>>S 500
```

Using the console commands, deposit this simple 11/780 program into location 500.

```
500/ MOVB #^XBC, R1  
      / HALT
```

Once you have executed your ODT program, R1 should contain BC.

3. Using ODT, write a program in machine (PDP-11) language that will rotate the LEDs continually on the 8232 module in the VAX 11/780 CPU.
4. Using console command language, set up the proper registers to stop the program in problem 3 to stop on micromatch with ROM state 62. Once the program stops on micromatch, examine V-bus channel 4 only.

BLANK

MICROCONTROL SUBSYSTEM

INTRODUCTION

This lesson will provide an introduction to the microsequencer hardware module. The entire theory of operation will not be covered within this module. The functional block diagram will be covered completely.

The microsequencer will be used as a focal point for the next lesson, where we will return to the microsequencer as required by the state of the CPU. As we progress, the full theory of the microsequencer will be presented.

This lesson will include how to read the microcode and interpret some fields contained within the microword (uword). Once again, not all the fields will be discussed during this lesson. The microword fields will be discussed as the logic requires and the entire microword will be covered within the next lesson.

The functional block diagram of PCS and WCS will be presented during this lesson. In-depth circuit analysis will be presented in the next lesson.

OBJECTIVES

1. Given a microcode macro, locate its definition in the PCS or WCS listing.
2. Given a microword address, locate that microword in the PCS or WCS listing.
3. Identify functions of the following functional logic:
 - a. Picosequencer
 - b. Microword multiplexer
 - c. Microstack
 - d. Compare register
4. Identify the functions of the following fields contained within the microword:
 - a. UJMP
 - b. BEN
 - c. USUB
5. Given a microword address, indicate whether it is located in PCS or WCS.

SAMPLE TEST ITEMS

1. The microcode macro "ALU_Q.MASK-1" has which component parts?
 - a. ALU/A-B-1,AMX/RAMX, RAMX/Q, BMX/MASK
 - b. ALU/A-B-1,AMX/RAMX, RAMX/MASK, BMX/Q
 - c. ALU/B-A-1,AMX/RAMX, RAMX/Q, BMX/MASK
 - d. ALU/B-A-1,AMX/RAMX, RAMX/MASK, BMX/Q

RESOURCES

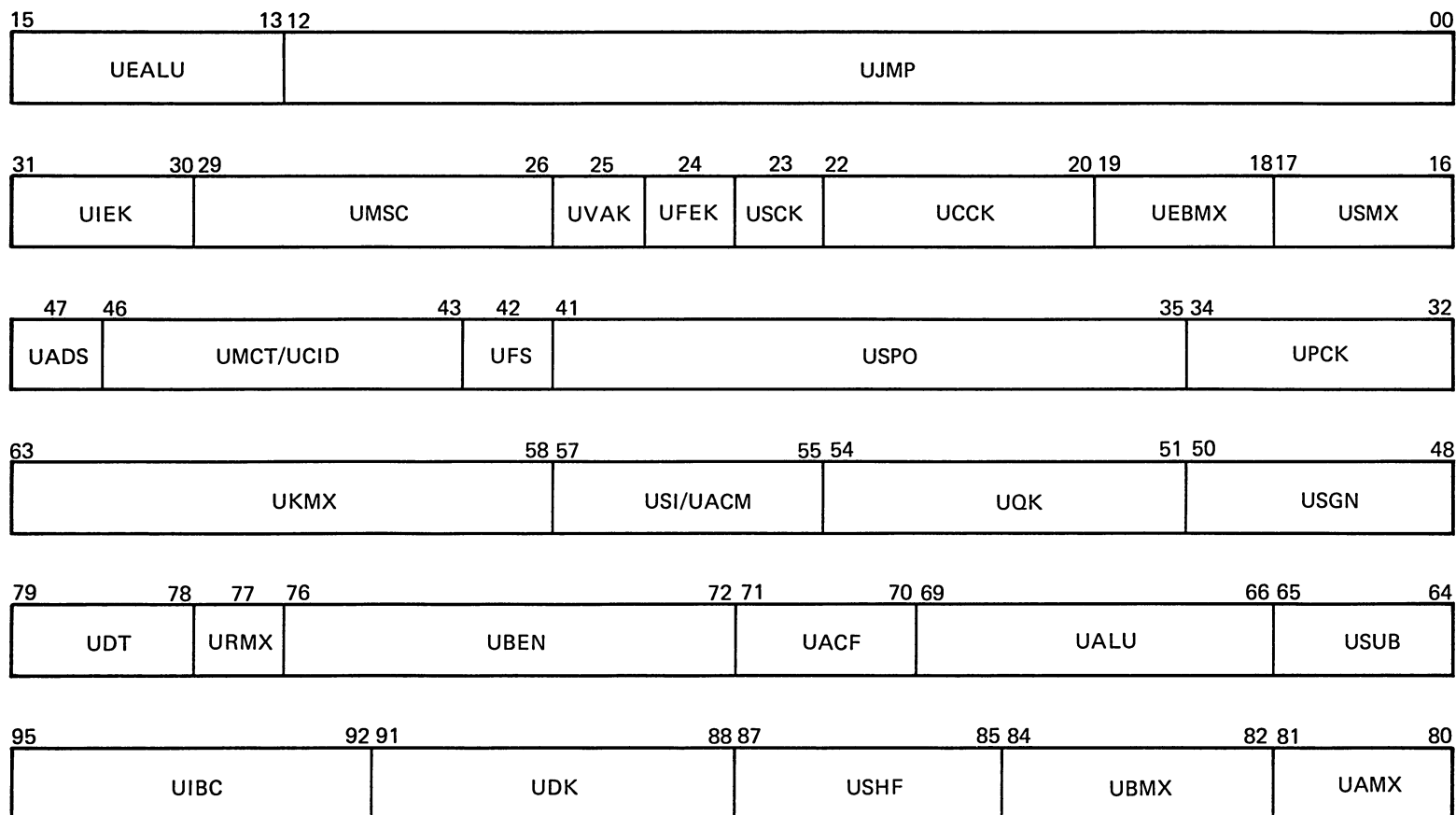
1. VAX-11/780 Print Set
2. VAX-11/780 System Maintenance Guide, Microword Field Specification Sheet, pp. 5-10-5-17
3. VAX-11/780 System Maintenance Guide, ID Bus Register Map, pp. 3-20-3-25
4. KA 780 Central Processor Technical Description

LECTURE OUTLINE

- I. Introduction
 - A. PCS
 - B. WCS
 - C. Microsequencer
 - D. Buses
- II. Programmable Control Store Address Logic
- III. Control Store Bus Parity
- IV. Microcontrol Subsystem Timing
- V. PCS Timing Sequence
- VI. Writable Control Store
- VII. WCS CS Write Logic
- VIII. Microsequencer: WCS Write Logic
- IX. Microprogram Counter Generation
- X. Microword Fields: Normal Mode
- XI. Microsubroutine Call and Return
- XII. Microsequencer: Normal Mode
- XIII. Microtrap Vectors
- XIV. Microtrap Flow
- XV. Microsequencer: Microtrap
- XVI. Micro-ECO Generation
- XVII. Micro-ECO Flow
- XVIII. Microsequencer: ECO Mode Logic
- XIX. Cache Stall Flow
- XX. Maintenance Mode
- XXI. Stop on Microbreak Match

LECTURE OUTLINE (Cont)

- XXII. Microsequencer: Maintenance Return
- XXIII. Initialize Mode



TK-1655

15	13 12	00
UEALU	UJMP	

<u>EXPONENT ALU</u>	NEXT MICROWORD ADDRESS
00 = A 01 = OR 02 = AND NOT 03 = B 04 = A + B 05 = A – B 06 = A + 1 07 = NABS A – B	

31	30 29	26	25	24	23	22	20 19	18 17	16
UIEK	UMSC	UVAK	UFEK	USCK	UCCK	UEBMX	USMX		

<u>INTER. AND EXCEP. ACK</u>	<u>MISCELLANEOUS</u>	<u>VA CNTR.</u>	<u>FE CNTR.</u>	<u>SC CNTR.</u>	<u>CONDITION CODES</u>	<u>EBMX SEL.</u>	<u>SMX SEL.</u>
00 = NOP 01 = ISTR 02 = IACK 03 = EACK	00 = NOP 01 = CHK.CHM 02 = CHK.FLT.OPR 03 = CHK.ODD.ADDR 04 = IRD 05 = LOAD.STATE 06 = LOAD.ACC.CC 07 = READ.RLOG 08 = CLR.FPD 09 = SET.FPD 0A = CLR.NEST.ERR 0B = SET.NEST.ERR 0C = SECOND REF 0D = RETRY.NO.TRAP 0E = RETRY.TRAP 0F = INH.CM.ADDR	0=NOP 1=LOAD	0=NOP 1=LOAD	0=NOP 1=LOAD	00 = NOP 01 = LOAD.UBCC 02 = SET.V 03 = TEST.Z 04 = ROR 05 = N + Z ← ALU 06 = C ← AMXO 07 = INST.DEP	00 = FE 01 = KMX 02 = AMX.EXP 03 = SHF.VAL	00 = EALU 01 = FE 02 = ALU 03 = ALU.EXP

TK-1654

47	46	43	42	41	35	34	32
UADS	UMCT/UCID			UFS	USPO		UPCK

<u>ADDR. SEL.</u>		<u>FUNC. SEL.</u>		SCRATCH PAD OPERATION (REFER TO TABLE 2-16)	<u>ADDRESS COUNT CONTROL</u> 00=NOP 01=PC←VA 02=PC←VIBA 03=VA+4 04=PC+1 05=PC+2 06=PC+4 07=PC+N
00=VA 00=IBA		00=MCT 01=CID			
		<u>UCID</u> <u>CONTROL AND ID BUS CONTROL</u> 01=NOP 05=ACK 07=CONT 09=READ SC 0B=READ KMX 0D=WRITE 5C 0F=WRITE KMX			
<u>UMCT</u> <u>MEMORY CONTROL</u> 00=TEST RCHK 02=MEM NOP 04=TEST WCHK 0A=WRITE.V.NOCHK 0C=WRITE.V.WCHK 0E=LOCKWRITE.V.XCHK 10=READ.V.RCHK 12=READ.V.NOCHK 14=READ.V.WCHK 16=READ.V.IBCHK 18=READ.V.NEWPC 1A=LOCKREAD.V.NOCHK 1C=LOCKREAD.V.WCHK				20=SBI HOLD 22=SBI.HOLD+UNJAM 24=INVALIDATE 26=VALIDATE 28=EXTWRITE.P 2A=WRITE.P 2E=LOCKWRITE.P 32=READ.P 36=READ.INT.SUM 3A=LOCKREAD.P 3E=ALLOW.IB.READ	

TK-1652

63	58 57	55 54	51 50	48
UKMX	USI/UACM	UQK	USGN	
<u>CONSTANTS SELECT</u> 00=8 01=1 02=2 03=3 04=4 05=SP1.CON 06=SP2.CON 07=SC 08 THROUGH 3F=CONSTANTS FROM SK ROM	<u>SHIFT INPUT CONTROL</u> 00=DIVD 01=ASHR 02=ASHL 03=ZERO 04=SPARE 05=DIV 06=MUL+ 07=MUL- <u>ACCEL MISC. CONTROL</u> 00=PWR.UP 01=ABORT 06=POLYDONE	<u>Q REG CONTROL</u> 00=NOP 01=LEFT 2 02=RIGHT 2 05=LEFT 06=RIGHT 08=SHF 09=SHF.FL 0A=DEC.CON 0B=ACCEL 0C=D 0E=ID 0F=CLR	<u>SIGN CONTROL</u> 00=NOP 01=LOAD SS 02=SS.FROM.SD 03=NOT.SD 04=SD.FROM.SS 05=SS.XOR.ALU 06=ADD.SUB 07=CLR.SD+SS	

TK-1653

Microword Format and Field Definitions (Sheet 4 of 5)

79	78	77	76	72 71	70 69	66 65	64
UDT	URMX	UBEN		UACF	UALU		USUB
<u>DATA TYPE</u> 00=LONG 01=WORD 02=BYTE 03=INST.DEP	<u>RMX SEL.</u> 00=D 01=Q	<u>BRANCH ENABLE</u> 00=NOP 01=Z 02=ROR 03=C31 06=ACCEL 08=DATA.TYPE 08=END.DP1 0A=REI 0A=SRC.PC 0B=IB.TEST 0C=MUL 0D=SIGNS 0E=INTERRUPT 0F=DECIMAL 10=UTRAP		<u>ACCEL. CONTROL</u> 00=NOP 01=SYNC 02=TRAP 03=CONTROL	<u>ALU CONTROL</u> 00=A-B 01=A-B.RLOG 02=A-B-1 03=INST.DEP 04=A+B+1 05=A+B 06=A+B.RLOG 07=A.OR.NOTB 08=A.XOR.B 09=A.AND.NOT.B 0A=NOTA 0B=A+B+PSL.C 0C=A.OR.B 0D=A.AND.B 0E=B 0F=A	<u>SUB-ROUTINE CONTROL</u> 00=NOP 01=CALL 02=RET 03=SPEC	

95	92 91	88 87	85 84	82 81	80
UIBC	UDK	USHF	UBMX	UAMX	
<u>INSTRUCTION BUFFER CONTROL</u> 00=NOP 01=STOP 02=FLUSH 03=START 04=CLR.0.1 05=CLR.2.3 07=BDEST 0C=CLR.O 0D=CLR.1 0E=CLR.0-3 0F=CLR.1-5 COND	<u>D REG CONTROL</u> 00=NOP 01=LEFT 2 02=RIGHT 2 04=DIV 05=LEFT 06=RIGHT 08=SHF 09=SHF.FL 0A=ACCEL 0B=BYTE SWAP 0C=Q 0D=DAL.SC 0E=DAL.SV 0F=CLR	<u>ALU SHIFTER CONTROL</u> 00=ALU 01=LEFT 02=RIGHT 03=ALU.DT 04=RIGHT.2 05=LEFT 3	<u>BMX SELECT</u> 00=MASK 01=PC.OR.LB 02=PACKED.FL 03=LB 04=LC 05=PC 06=KMX 07=RBMX	<u>AMX SEL.</u> 00=LA 01=RAMX 02=RAMX.SXT 03=RAMX.OXT	

TK-1656

BLANK

EXERCISES

These exercises, consisting of four microword worksheets, will help your understanding of the microword, micro-macros, micro-macro expansion, and decoding and encoding the VAX-11/780 microword.

Later, while studying the WCS Debugger, you will use the ten blank worksheets supplied here as a training aid.

You are not expected to understand everything you read at this point. This section will serve only as an introduction to the material you will work with. But by the end of the course, you should fully understand all the concepts presented here. So give it some time.

First, read pages 2-8 through 2-12 of the KA780 Central Processor Technical Description, "How to Read the Micro Code." This same information (except for a few minor differences) can be found in the System Maintenance Guide starting on page 5-7. The information as it appears in the System Maintenance Guide will also be found on the ESOAB microfiche.

Now, look at Microword 1 Worksheet for the first example.

In the upper left-hand corner, you will see a series of boxes labeled Microword in Hex. (In this example, they are left blank; later, they will be used.) Just under this is the term MACROS and on this line is:

R(SC)_Q

This is a micro-macro expression. The definition is found on page 2-10 of the KA780 Central Processor Technical Description, but let's look at it again.

A .macro is a symbol whose value is one or more field/value specifications. A macro definition is a line containing the macro name followed by a quoted string, which is the value of the macro (macro expansion).

At the bottom of the microword worksheet are lines that are designated Macro Expansion. This macro expansion is gained from either of two sources:

1. System Maintenance Guide starting on page 5-19 and continuing through page 5-43.
2. ESOAB microfiche. This has been reproduced in the System Maintenance Guide.

EXERCISES

The macro expansion on the microword worksheet is located on page 5-36 in the System Maintenance Guide. After you have located the expression expansion on that page, survey the pages that contain all the macros and their expansions. Notice that for the most part the list is in alphabetical order.

Next, put the associated hex values in binary format within the microword. This is a very simple procedure to follow. Let's do the first one:

RAMX/Q

For this expression, we must locate the RAMX field within the control ROM field definitions which are on pages 5-10 through 5-17 in the System Maintenance Guide. Once again, the fields are in alphabetical order.

Page 2-8 of the KA780 Central Processor Technical Description shows microcode field definitions having the form

SYMBOL/=J,K,L,M

The RAMX field is on page 5-15. Let's look at that.

```
RAMX/=0,1,77,D           ;DATA PATH MIXER TO AMX
      D=0                 ;DEFAULT
      Q=1
```

Here, the symbol is RAMX. The J parameter is only meaningful when D is specified as the default mechanism. In the example, D is present in the definition; thus, the 0 means the default value for that expression.

In contrast, look at the KMX definition on page 5-13 of the System Maintenance Guide and notice there is no D in the expression.

Once again, the D tells the micro-macro assembler to force a default value in that field of the microword. Later, other examples will be given.

The K parameter defines the field size in the number of bits (in decimal), and in this example, RAMX field is only one (1) bit in length.

The L parameter defines the field position (in decimal) as the bit number of the right-most bit of the field, in this example, bit position 77.

EXERCISES

Let's move on to the second expansion:

AMX/RAMX

The binary value for this expansion is found on page 5-10 of the System Maintenance Guide. Again, notice the binary value placed in the AMX field of the microword is one (1) and the AMX has no default value assigned.

The next expansion is:

ALU/A

The definition is found on page 5-10 of the System Maintenance Guide. The binary value assigned is F (1111).

Continue by placing the remaining binary values in the microword fields. Do not concern yourself yet with fields not within the expansions. After completing that, look at fiche ESQAB-1, 4 of 4, frame number N5, line number 2709 where this expression is found on line number 24668 within this listing. At line number 2704, the micro-macro R(SC)_ALU is found in many microwords versus the R(SC)_Q which is found in only one microword.

If you locate line number 24668 on ESQAB-1, 3 of 4, frame L16, you will see:

R(SC)_Q, LONG
J/CONS.HALT

The LONG states the data type (DT). The definition for this field is found on page 5-11 of the System Maintenance Guide. The binary value is 0, which you can verify by locating the DT field within the microword value given on the fiche, as well as all other values that you did on your worksheet. Let's discuss another entry in this microword.

J/CONS.HALT

EXERCISES

On page 2-9 of the KA780 Central Processor Technical Description Paragraph 2.2.1.3 (Label Definitions) you will see that a microinstruction may be labeled by a symbol followed by a semicolon preceding the microinstruction definition. In this example, the format is slightly different, but the concept is the same. The KA780 Central Processor Technical Description also states that the address of the microinstruction becomes the value of the symbol in the field named J. In this example, CONS.HALT is the symbol, and the UJMP field will get a value of 439 by the microassembler. Locate micro 439 on line number 24710, the next frame down from L16. This microword has a label CONS.HALT assigned.

The next two worksheets contain micro-macro expressions and you will:

1. Locate and write the macroexpansions on the worksheets.
2. Write in the binary values.
3. Locate the line numbers of the microwords that these expressions exist.

After you have completed these worksheets, refer to the information sheet for further discussion.

On the next worksheet, you will find the blocks labeled "Microword in Hex" filled in with microword values. Decode these values to their respective ROM field definitions as given on pages 5-10 - 5-17 of the System Maintenance Guide.

Remember, you will not understand all the definitions yet, but this exercise will familiarize you with decoding and encoding the microword found in the VAX-11/780. Check with the instructor if you have any problems with the exercise.

The next exercise will amplify a portion of Paragraph 2.2.2.7 in the KA780 Central Processor Technical Description page 2-10.

Macros may have parameters enclosed in square brackets ([and]). The definition of a macro with parameters includes paired brackets to indicate where the parameters should go. It uses "@" followed by a decimal digit string to indicate which symbols in the macro body should be replaced by the parameters. The symbol "@" will not be found in the microfiche listing.

EXERCISES

The best way to approach parameters is by working an example. As a random choice by the author, refer to page 5-35 of the System Maintenance Guide. The expression chosen is:

```
RC[]_K[]
```

Using this expression, refer to microfiche ESQAB-1, 4 of 4, frame B6, line number 2650. At this line number you will find a series of line numbers where this expression can be found. For our example, let's use line numbers 3399 and 3419 which are found on fiche 1 of 4. Line number 3399 is located on frame K7.

```
RC T0_K ZERO
```

Notice there are no "@" signs in the expression found in the listing. Simply, since T0 is first, it will be assigned to @1; and since ZERO is second, it will be assigned to @2. (Appearance of order is left to right.) If you go back to the expression on page 3-35 of the System Maintenance Guide, you can fill in the expression.

```
"KMx/ZERO _____ SPO.PC/T0"
```

If you refer again to page 2-10 and re-read Paragraph 2.2.1.7 in the KA780 Central Processor Technical Description, the example should make more sense now.

Finally, here is a brief introduction of the relationship between values found in the microword and their impact on the data paths. This will give you an idea of their function and will help you when you write and execute your own microwords.

Refer to the block diagram for data path control M8229. At the same time, let's use a simple expression for analysis:

```
D_Q-D "RAMX/Q,AMX/RAMX,RBMX/D,BMX/RBMX,ALU/A-B,SHF/ALU,DK/SHF"
```

This expression is found on page 5-27 of the System Maintenance Guide. Once the assembler has encoded this macro, the microword contains all the proper values to manipulate the data path. By looking at the expansion and then the block, you can visualize the data flowing through the logic. Just remember that the values in the microwords are enabling levels for the logic.

[illegible]

R(SC)–Q

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
IEK		MSC				VAK	FEK	SCK		CCK		EBMX		SMX		EALU				JMP											

[illegible]

95	94	93	92	91	90	89	88	87	86	85	84	83	82	81	80	79	78	77	76	75	74	73	72	71	70	69	68	67	66	65	64		
IBC				DK				SHF				BMX				AMX		DT		RMX		BEN				ACF		ALU				SUB	

RAMX/Q AMX/RAMX, ALU/A, SHF/ALU SPO AC/WRITE, RAB, SPO ACN/SC

[illegible]

ALU-OTMASK + 1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
IEK		MSC				VAK FEK		SCK		CCK		EBMX		SMX		EALU		JMP													

ACM

Si

1
X

CID

MCT

95	94	93	92	91	90	89	88	87	86	85	84	83	82	81	80	79	78	77	76	75	74	73	72	71	70	69	68	67	66	65	64		
IBC				DK				SHF				BMX				AMX		DT		RMX		BEN				ACF		ALU				SUB	

[illegible]

ALU_D+Q+PSL.C

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
IEK		MSC				VAK	FEK	SCK	CCK			EBMX		SMX		EALU				JMP											

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32	
KMX								QK				SGN				ADS				FS	SPO						PCK					
						ACM		SI										X	CID		MCT											

95	94	93	92	91	90	89	88	87	86	85	84	83	82	81	80	79	78	77	76	75	74	73	72	71	70	69	68	67	66	65	64				
IBC				DK				SHF				BMX				AMX		DT		RMX		BEN						ACF		ALU				SUB	

MACRO EXPANSION

4-19

Microcontrol Subsystem

4-19

Microcontrol SubsystemMicrocontrol SubsystemMicrocontrol Subsystem

[illegible]

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
IEK		MSC				VAK	FEK	SCK	CCK			EBMX		SMX		EALU				JMP											

4-20Microcontrol Subsystem

[illegible]

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
IEK		MSC				VAK	FEK	SCK	CCK			EBMX		SMX		EALU				JMP											

MCTSUB

[illegible]

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
IEK		MSC				VAK	FEK	SCK	CCK		EBMX	SMX		EALU		JMP															

Si

MCT

[illegible]

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
IEK		MSC				VAK FEK		SCK		CCK		EBMX		SMX		EALU				JMP											

ACM SI X CID MCT

[illegible]

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
IEK		MSC				VAK FEK		SCK		CCK		EBMX		SMX		EALU				JMP											

MCT

SUB

[illegible]

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
IEK		MSC				VAK	FEK	SCK	CCK		EBMX		SMX		EALU				JMP												

MCTSUB

[illegible]

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
IEK		MSC				VAK	FEK	SCK	CCK			EBMX		SMX		EALU			JMP												

15

95	94	93	92	91	90	89	88	87	86	85	84	83	82	81	80	79	78	77	76	75	74	73	72	71	70	69	68	67	66	65	64				
IBC				DK				SHF				BMX				AMX		DT		RMX		BEN						ACF		ALU				SUB	

4-27

MACROS

MICROWORD IN BINARY

MACRO EXPANSION

[illegible]

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
IEK		MSC				VAK	FEK	SCK	CCK			EBMX		SMX		EALU				JMP											

ACM SI X CID MCT

95	94	93	92	91	90	89	88	87	86	85	84	83	82	81	80	79	78	77	76	75	74	73	72	71	70	69	68	67	66	65	64				
IBC				DK				SHF				BMX				AMX		DT		RMX		BEN						ACF		ALU				SUB	

MICROWORD IN HEX

--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--

MACROS

MICROWORD IN BINARY

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
IEK		MSC				VAK	FEK	SCK	CCK				EBMX		SMX	EALU				JMP											

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
KMX						OK						SGN				ADS					FS	SPO								PCK	
																	X		CID		MCT										

95	94	93	92	91	90	89	88	87	86	85	84	83	82	81	80	79	78	77	76	75	74	73	72	71	70	69	68	67	66	65	64
IBC				DK				SHF				BMX		AMX		DT	RMX	BEN				ACF		ALU				SUB			

MACRO EXPANSION

BLANK

MODULE TEST

1. Given a list of microword addresses, indicate these items for each:

- a. Found in which listings PCS (P) or WCS (W).
- b. Fiche and frame number

Address	P or W	Fiche	Frame No.
0BF2			
0F8D			
1170			
11A6			
04B3			

2. The microcode macro "ALU_Q-MASK-1" has which component parts?
- a. ALU/A-B-1, AMX/RAMX, RAMX/MASK, BMX/Q
 - b. ALU/A-B-1, AMX/RAMX, RAMX/Q, BMX/MASK
 - c. ALU/B-A-1, AMX/RAMX, RAMX/MASK, BMX/Q
 - d. ALU/B-A-1, AMX/RAMX, RAMX/Q, BMX/MASK
3. The last microinstruction of a microtrap handler routine would contain a USUB field of
- a. 0
 - b. 1
 - c. 2
 - d. 3
4. Which microword field controls whether the instruction decode logic will affect the UPC bus bits?
- a. UBEN
 - b. UDT
 - c. UJUMP
 - d. USUB

MODULE TEST

5. During normal mode of the picosequencer, bit 12 of the UPC address can be controlled by all but the
 - a. UJMP field
 - b. FPA uMUX
 - c. NUA bus
 - d. CIBN UPC 12
6. Which of the following conditions would not cause the microstack pointer to increment/decrement and the UPCSV address popped/pushed on the stack?
 - a. CALL (USUB=1)
 - b. RETURN (USUB=2)
 - c. Decision point branch
 - d. Micro-ECO mode
7. The picosequencer selects the output from the micromultiplexer depending on
 - a. a microcode command
 - b. a macrocode command
 - c. the priority of the function to be performed
 - d. the branch enabled selected
8. Microtrap vectors are input to the next microaddress bus from the
 - a. UECO register
 - b. UBEN inputs
 - c. ID bus
 - d. CIB
9. The field programmable logic array (FPLA) contains
 - a. hardware ECO information
 - b. microcode EECO data
 - c. UPC address of ECO data
 - d. abort address in WCS
10. During a microbreak match function, a sync pulse is produced
 - a. at CPT1 of the chosen microstate
 - b. in CPT3 of the preceding microstate
 - c. at CPT0 of the chosen microstate
 - d. no sync pulse is generated

11/780 INTERNAL OPERATIONS

INTRODUCTION

This lesson is an intricate analysis of the internal logic down to chip level. The lesson will be supplemented with an extensive laboratory exercise to help you develop the practical skills necessary to apply your theoretical knowledge.

The lecture/lab focuses on a selected number of ROM states that are most commonly used to execute a process. The discussion of these ROM states will lead you through 75 percent of the total print set.

5

OBJECTIVES

For each of the events listed below:

1. List the first three microword addresses in PCS sequence, and
2. Identify the determining functional logic within the CPU print set that creates the entry point into PCS.
 - a. Translation buffer hit/miss
 - b. Instruction buffer hit/miss
 - c. Page table length violation
 - d. Page table entry access violation
 - e. Invalid page table entry violation
 - f. System initialization
 - g. Cache stall
 - h. Instruction buffer stall
 - i. Any given VAX macroinstruction
 - j. External interrupt
 - k. Internal interrupt

- l. Control store parity error
- m. Odd address error
- n. Timeout
- o. Read data substitute
- p. Cache parity error
- q. Translation buffer parity error
- r. Modify bit
- s. Page trap
- t. Unalign trap

SAMPLE TEST ITEMS

Match the entry points (Column A) of PCS with the correct events (Column B) for that entry point into PCS.

Column A

- 1. ____ 105
- 2. ____ 060
- 3. ____ 062
- 4. ____ 127
- 5. ____ 038

Column B

- A. Instruction buffer miss
- B. IRD state
- C. Control store parity error
- D. System initialization
- E. Translation buffer miss

RESOURCES

- 1. KA780 Print Set
- 2. VAX-780 Microfiche Library
- 3. VAX-11/780 Central Processor Technical Description
- 4. VAX-11/780 TB/CACHE/SBI Control Technical Description

LECTURE OUTLINE

- I. Introduction
- II. ROM states required to execute program under study

A. 0127	AA. 0B79
B. 0FDB	BB. 0E8A
C. 03DF	CC. 0E8C
D. 00AB	DD. 04AF
E. 0062	EE. 0B61
F. 0060	FF. 0E68
G. 0E64	GG. 0E69
H. 0E66	HH. 0062*
I. 0B60	II. 0009
J. 023F	JJ. 0094
K. 045F	KK. 0224
L. 0E7D	LL. 0105
M. 0E80	MM. 0004
N. 0FA7	NN. 02C4
O. 0E81	OO. 008F
P. 0E84	PP. 039F
Q. 025F	QQ. 00FF
R. 0E85	
S. 0B78	
T. 0E94	
U. 0E96	
V. 0FAF	
W. 0E98	
X. 0E99	
Y. 0E9A	
Z. 04CF	
- III. Instruction decode logic
 - A. IRCC ROM listing
 - B. Instruction decode logic
 - C. Instruction data path
- IV. Exception and interrupt logic
 - A. Trace an interrupt through the logic
 - B. Show microcode relating to ROM state

LECTURE OUTLINE

- I. Comments from the PCS listing for 127
 - A. Continue
 - B. Clear instruction buffer
 - C. Test for low power
- II. Types of starts
 - A. Console
 - 1. Start
 - 2. Continue
 - B. Load context instruction
- III. Key logic or signals
 - A. Flush
 - B. Low power test
- IV. Discussion questions
 - A. Notice that the VA register is loaded by the VAK field. What will be the contents of this register at the end of this 200 nanosecond time slice?
 - B. The IBC field loads the virtual instruction buffer address (VIBA). What signal will load the VIBA?
 - C. Which fields of this microword are at their default value?

LECTURE OUTLINE

- I. Comments from the PCS listing for FDB
 - A. Start IB
 - B. Test compatibility mode
- II. Key signals or logic
 - A. Start IB
 - B. Stop IB
 - C. BEN field
- III. Discussion questions
 - A. On what logic page is the logic that checks for CM or IS or kernel mode?

LECTURE OUTLINE

- I. Comments from PCS listing for 3DF
 - A. Clear console mode
- II. Key logic or signals
 - A. Clear console mode
 - B. Console ACK
 - C. Console attention light
- III. Discussion question
 - A. The signal CIBN ATTN H (CIBA,SCPA) [3,C] implies that this signal will go to the system control panel. How?
 - B. How is the console command flip-flop on M8231 (ICLF[3,B]) affected by ROM state 00FF?

LECTURE OUTLINE

- I. Comments from PCS listing for AB
 - A. Skip PC past op code of new instruction
 - B. Fill IB with new instruction
- II. Key signals or logic
 - A. Increment PC
 - B. MCT field
 - C. VA register
 - D. Odd address trap
- III. Discussion question
 - A. Why is the signal TBMC ENABLE IA H Low (DEPN[6,A]) during this ROM state?
 - B. Have the student note on the chart for MCT field in the System Maintenance Guide, page 5-18, why only ODDADDRESS can create a microtrap condition.

LECTURE OUTLINE

- I. Comments from the PCS listing for 62
 - A. Waiting for data
 - B. Loop on it
- II. Key signals or logic
 - A. MCT field
 - B. SUB field
 - C. IRCH NO FAULT H
 - D. IRCH ERR H
 - E. Service bits
 - F. IRCE SRV + INV L
 - G. IRCE STAL + SVC H
- III. Discussion questions
 - A. If memory management is not turned on, and IDPJ B0 VAL (0) H was high, what address will be passed from this multiplexer?

LECTURE OUTLINE

- I. Comments from the PCS listing for 60
 - A. Stopped with TB miss
 - B. Go fill TB
- II. Key signals or logic
 - A. IB miss
 - B. TB miss
 - C. SUB field
 - D. Microstack
- III. Discussion questions
 - None

LECTURE OUTLINE

- I. Comments from the PCS listing for E64
 - A. Set Nested Error flag
 - B. Save PC in RC register
 - C. Get VIBA
 - D. Save contents of D-register
 - E. Move contents of Q-register to D-register
- II. Key signals or logic
 - A. SHF sets ALU
 - B. ALU sets BMX
 - C. BMX set PC
 - D. RC registers
 - E. ID addresses
 - F. Set VIBA
 - G. Nested Error flag
 - H. FAULT

The instruction is faulted if the machine check is not a control store parity error or a second machine check, or is not in memory management or interrupt/exception firmware when the machine check occurs.
 - I. ABORT

An instruction is aborted if it is in memory management or interrupt/exception firmware when the check occurs and on a control store parity check.
 - J. HALT

The instruction is halted if a machine check is detected as having occurred during the error handling firmware, or on a double SBI CP error noted in the ID SBI.ERR<2>. The machine is always halted if the processor is in console mode when the machine check occurs.
- III. Discussion questions
 - A. How is the RC register written into?
 - B. Where are the internal registers located and how is it written?

LECTURE OUTLINE

- I. Comments from the PCS listing for E66
 - A. Move contents of VIBA to both VA and PC registers
 - B. Save contents of Q-register
- II. Key signals or logic
 - A. Q.SV register
- III. Discussion questions
 - A. RC C contains?
 - B. PC contains?
 - C. VA contains?
 - D. ID 2E contains?
 - E. ID 2F contains?

LECTURE OUTLINE

- I. Comments from the PCS listing B60
 - A. Do not allow IPA refill
 - B. Save entry contents of SC register
 - C. Go get PTE (Page Table Entry)
- II. Key signals or logic
 - A. IPA register
 - B. MCT field
 - C. SUB field
 - D. BEN field
 - E. TBMC AR FLOP (1) L
- III. Discussion questions
 - None

LECTURE OUTLINE

- I. Comments from the PCS listing for 23F
 - A. Do not allow the IPA refill
 - B. Not compatibility mode
 - C. Start extraction of VPN from the VA and decode address space type
- II. Key signals or logic
 - A. Extraction of VPN from VA
 - B. Test for address region
 - C. Test for compatibility mode
- III. Discussion question
 - A. On what logic print is the determination made for address space?

LECTURE OUTLINE

I. Comments from the PCS listing for 45F

- A. P0 space address
- B. Save VA
- C. Right adjust VPN
- D. Get P0LR into Q-register

II. Key signals or logic

A. Shifters

1. Level 1 (DBPN)

S1	S2	
0	0	No shift
0	1	16-bit shift
1	0	32-bit shift
1	1	48-bit shift

2. Level 2 (DBPP)

S1	S0	
0	0	No shift
0	1	4-bit shift
1	0	8-bit shift
1	1	12-bit shift

3. Level 3 (DBPS)

S1	S0	
0	0	No shift
0	1	1-bit shift
1	0	2-bit shift
1	1	3-bit shift

B. UDK field

C. P0 length register

III. Discussion questions

Class Project

To fully appreciate the value of this, let the student perform the actual shift as described by the microword. The only assumption that will be made is that the address the VPN is being extracted from is not zero. The number for this example will have a one (1) in bit position 10 of the D-register prior to the shift and the shift counter has FFF6 contained.

LECTURE OUTLINE

- I. Comments from the PCS listing for E7D
 - A. Set up test for length violation
 - B. Do not allow IPA refill
- II. Key signals or logic
 - A. Set up test for length violation
 - B. PSL register

III. Discussion questions

Class Project

An interesting project would be to trace the setting of a bit in the PSL.

LECTURE OUTLINE

- I. Comments from the PCS listing for E80
 - A. Shift VPN left 2
 - B. Load P0BR into Q-register
 - C. Test for length violation
- II. Key signals or logic
 - A. Simple rules for P0 space
 - 1. If the result is positive, more pages referenced than length allocated.
 - 2. If the result is negative, more length allocated than pages referenced.
 - B. BEN field
 - C. Double shift left of D-register
 - D. Test length violation
- III. Discussion questions
 - A. What is the address of the next micro-PC if there was a length violation?

LECTURE OUTLINE

- I. Comments from the PCS listing for FA7
 - A. No length violation
 - B. Calculate PxPTE virtual address
- II. Key signal or logic
 - A. Formation of system virtual address
 - B. BEN field
- III. Discussion questions
 - None

LECTURE OUTLINE

- I. Comments from the PCS listing for E81
 - A. Set up test for PxPTE's virtual address and save it.
- II. Key signals or logic
 - A. Where address of PxPTE is saved
 - B. How the test for PxPTE is performed in the TB
 - C. What impact on cache does this test have?
 - D. What impact on the SBI does this test have?
 - E. What impact on the translation buffer does it have?
- III. Discussion questions
 - None
 - An interesting project would be to let the student trace the logic involved with this test to find out how it is performed.

LECTURE OUTLINE

- I. Comments from the PCS listing for E84
 - A. Test PxPTE virtual address
 - B. Inhibit CM address mode
 - C. Do not allow IPA refill
 - II. Key signals or logic
 - A. BEN field
 - III. Discussion questions
- Class Project
- The students should trace the BEN logic to create the micro address 25F.

LECTURE OUTLINE

- I. Comments from the PCS listing for 25F
 - A. PxPTE not in translation buffer
 - B. Do not allow IPA refill cycles
 - C. PTE.VA to both PC and VA registers
- II. Key signals or logic
 - A. VAK field
 - B. PCK field
- III. Discussion questions
 - None

LECTURE OUTLINE

- I. Comments from the PCS listing for E85
 - A. Start extracting VPN from PTE.VA
 - B. Do not allow IPA refill cycles
 - C. Clear PTE.VA<31>
 - D. Clear Q-register
 - E. Set up to right adjust VPN
- II. Key signals or logic
 - A. Clear bit <31> PTE.VA
 - B. DK field
 - C. Extraction of VPN from PTE.VA
- III. Discussion questions
 - None

LECTURE OUTLINE

- I. Comments from the PCS listing for B78
 - A. System length register to Q
 - B. VPN to D-register right-adjusted
 - C. Merge with system PTE fetch routine
- II. Key signals or logic
 - A. Loading of Q-register
 - B. SUB field
- III. Discussion questions
 - A. There have been three pushes on the microstack.
What addresses have been pushed on?

LECTURE OUTLINE

- I. Comments from the PCS listing for E94
 - A. Do not allow IPA refill
 - B. Set up for length violation test
- II. Key signals or logic
 - A. Q-register gets difference from D-Q
- III. Discussion questions
 - None

LECTURE OUTLINE

- I. Comments from the PCS listing for E96
 - A. Shift VPN left 2
 - B. Load SBR into Q-register
 - C. Test for length violation
- II. Key signals or logic
 - A. Multiply VPN by 4
 - B. BEN field
- III. Discussion questions
 - A. If the results are positive when the BEN field tests for signs, there is a length violation. Locate the logic that would NOT set this bit.

LECTURE OUTLINE

- I. Comments from the PCS listing for FAF
 - A. Passed length test
 - B. Do not allow IPA refill cycles
 - C. Calculate SPTE physical address and load into
 - 1. Virtual address register
 - 2. RC<PTE.PA>
- II. Key signals or logic
 - A. RC register
 - B. VA register
 - C. Q-register
- III. Discussion questions
 - None

LECTURE OUTLINE

- I. Comments from the PCS listing for E98
 - A. Fetch SPTE into D-register
 - B. Inhibit CM address format
- II. Key signals or logic
 - A. SBLT STALL L - The cache stall logic is on SBI interface low bits (8218) located at SBLT [2,C]. The signal SBLT STALL L will create a stall condition when LOW. That also tells you that if any of those inputs are low, the stall logic is not invoked. To prove a stall condition all those input gates must not be enabled.

This project could last forever! Take my word: at this time VAX will stall to perform an SBI cycle.
 - B. SBI cycle
- III. Discussion questions
 - 1. Trace the logic required to load the D-register during this ROM state.
 - 2. Trace the logic necessary and required to place the high address of the PA out on the SBI.
 - 3. Using the stall charts found in the print set, justify a stall condition.

LECTURE OUTLINE

- I. Comments from the PCS listing for E99
 - A. Original PTE to Q-register
 - B. Do not allow IPA refill cycles
 - C. Mask out MBZ bits
- II. Key signals or logic
 - A. Mask out the MBZ bits
 - B. RC group register F contains mask
- III. Discussion questions
 - Student Project
 - Locate line 8324 in the PCS to confirm MASK.

LECTURE OUTLINE

- I. Comments from the PCS listing for E9A
 - A. Do not allow IPA refill cycles
 - B. Restore SVA into VA
 - C. Test SPTE for
 - 1. Validity
 - 2. No access
 - 3. MBZ field
 - II. Key signals or logic
 - A. Locate original SVA
 - B. BEN field
 - III. Discussion questions
- Class Project
- 1. Locate the logic that will modify the next microword address from 4CF to 4DF.
 - 2. Locate the logic for SPTE INVALID. Is it any different than that in question number 1?

LECTURE OUTLINE

- I. Comments from the PCS listing for 4CF
 - A. SPTE valid
 - B. Load SPTE into TBUF
- II. Key signals or logic
 - A. Load translation buffer
 - B. TBMD TB WRITE ENABLE H
 - C. SUB field
- III. Discussion questions
 - None

LECTURE OUTLINE

- I. Comments from the PCS listing for B79
 - A. Return from SPTE fetch routine
 - B. Fetch PxPTE
 - C. Merge into PxPTE fetch routine
- II. Key signals or logic
 - A. Fetching P0PTE
 - B. Load the D-register
- III. Discussion questions
 - None

LECTURE OUTLINE

- I. Comments from the PCS listing for E8A
 - A. Original PTE to Q-register
 - B. Do not allow IPA refill cycles
 - C. Mask out MBZ bits
- II. Key signals or logic
 - A. What is the original PTE?
 - B. Mask out MBZ bits
- III. Discussion questions
 - None

LECTURE OUTLINE

- I. Comments from the PCS listing for E8C
 - A. Restore VA register to original contents
 - B. Do not allow IPA refill cycles
 - C. Test PxPTE for
 - 1. Validity
 - 2. Not no acess
 - 3. MBZ field
- II. Key signals or logic
 - A. BEN field
- III. Discussion questions
 - Class Project
 - 1. Locate the logic that will force the microcode to 4BF instead of 4AF.
 - 2. For what use will the output of the ALU be in microword E8E?

LECTURE OUTLINE

- I. Comments from the PCS listing for 4AF
 - A. PxPTE is OK
 - B. Load PxPTE into translation buffer
 - C. Return to calling routine
- II. Key signals or logic
 - A. Load the translation buffer
 - B. SUB field
- III. Discussion question
 - None

LECTURE OUTLINE

- I. Comments from the PCS listing for B61
 - A. Restart IBUF
 - B. Restore SC to entry contents
 - C. Do translation and get next IBUF longword
- II. Key signals or logic
 - A. Start IB
 - B. MCT field
 - C. IPA
- III. Discussion questions
 - None

LECTURE OUTLINE

- I. Comment from the PCS listing for E68
 - A. Restore PC to entry value
- II. Key signals or logic
 - A. Where does VAX locate entry value of PC?
- III. Discussion questions
 - None

LECTURE OUTLINE

- I. Comments from the PCS listing for E69
 - A. Restore D to entry contents
 - B. Clear Nested Error flag
 - C. Restore Q to entry contents
 - D. Return to caller
- II. Key signals or logic
 - A. Clear Nested Error flag
 - B. SUB field
- III. Discussion questions
 - None

LECTURE OUTLINE

- I. Comments from the PCS listing for 62 (IRD)
 - A. Waiting for data
 - B. Loop on it
- II. Key signals and logic
 - A. Picosequencer
 - B. Cache stall
 - C. USCM UPC LOOP (0) H
 - D. USCL CALL 03 L
 - E. USCM IBUF EN (07:00) L
- III. Discussion questions
 - None

LECTURE OUTLINE

- I. Comments from the PCS listing for 009
 - A. (R)+ autoincrement
 - B. Use unincremented address in D and VA
- II. Key signals or logic
 - A. R-log
- III. Discussion questions
 - None

LECTURE OUTLINE

- I. Comments from the PCS listing for 94
 - A. Get here by data type
 - B. Normal B,W,L or F data
 - C. Go evaluate the second specifier
- II. Key signals or logic
 - A. MCT field
- III. Discussion questions
 - None

LECTURE OUTLINE

- I. Comments from the PCS listing for 224
 - A. Write to register
 - B. Store results in register
 - C. Set condition code from it
 - D. Go to next instruction
- II. Key signals or logic
 - A. Microtrap logic
 - B. USCB UTRAP H
 - C. CEHE UTRAP L
 - D. TBMW EN UNALIGN TRAP H
 - E. TBMW WOMBAT H
 - F. TBMW AARDVARK L
- III. Discussion questions
 - None

LECTURE OUTLINE

- I. Comments from the PCS listing for 105
 - A. Stop IBUF memory references
 - B. Save D register in ID temporary
 - C. Save "PC" in RC register set
 - D. Move D to Q-register

- II. Key signals or logic

There really is no discussion at this time. The logic should be obvious. ROM states 105,E42 and B48 are microstates set before merging with flows fully discussed in details starting at 023F. For any other, refer to microfiche for comments and contents of microword.

Read the description found in the fiche library relating to "Memory Management Firmware Description" ESOAB fiche 3 of 4, frame D12.

BLANK

SUPPLEMENT A – VAX-11/780 MICROSEQUENCES

Below is a listing of all the microstates that the VAX-11/780 needs to perform the program under study for this course. The only stipulation is that cache and the translation buffer are loaded with all the required data.

0127
0FDB
03DF
00AB
0062
0009
0094
0224
0062
0009
0094
0224
0062
0004
02C4
0062
008F
039F
00FF
00FF

Figure 5-1. Normal flow with cache and translation buffer loaded with all valid data.

Below is the microsequence necessary to perform the process under study. During this sequence the cache and the translation buffer were invalidated at the beginning.

0127	04CF	0E81	0E84
0FDB	0B79	0E84	025C
03DF	0E8A	025C	0E8A
00AB	0E8A	0E8A	0E8A
0062	0E8A	0E8C	0E8A
0060	0E8A	04AF	0E8A
0E64	0E8A	0B49	0E8A
0E66	0E8A	0D6C	0E8A
0B60	0E8A	0B4B	0E8A
023F	0E8A	0E48	0E8A
045F	0E8C	0D1C	0E8C
0E7D	04AF	0224	04AF
0E80	0B61	0224	0B49
0FA7	0E68	0224	0D6C
0E81	0E69	0224	0B4B
0E84	0062	0224	0E48
025F	0062	0224	0D1C
0E85	0062	0224	0224
0B78	0062	0224	0224
0E94	0062	0062	0224
0E96	0062	0009	0224
0FAF	0009	0094	0224
0E98	0094	0224	0224
0E99	0224	0105	0224
0E99	0105	0E42	0224
0E99	0E42	0B48	0062
0E99	0B48	023F	0004
0E99	023F	045F	02C4
0E99	045F	0E7D	0062
0E99	0E7D	0E80	008F
0E99	0E80	0FA7	039F
0E9A	0FA7	0E81	00FF

Figure 5-2. Microsequence for process under study.

This series of microsequences analyzes various violations in P0 space and their impact on the microsequence.

Normal Flow	P0PTE Invalid	Access Violation
0127	0127	0127
0FDB	0FDB	0FDB
:	:	:
:	:	:
0E99	0E99	0E99
:	:	:
:	:	:
0E99	0E99	0E99
0E9A	0E9A0E99	
0E9A	0E9A	0E9A
04CF	04CF	04CF
0B79	0B79	0B79
0E8A	0E8A	0E8A
:	:	:
:	:	:
0E8A	0E8A	0E8A
0E8C	0E8C	0E8C
04AF*	04BF**	04BF**

* Microstate 04AF is the normal entry point.

** Microstate 04BF is the entry point for both violations.

Figure 5-3. Microflow study for violations in P0 space.

This series of microsequences analyzes various violations in system space and their impact on the microsequence.

Normal Flow	SPTE Invalid	Access Violation
0127	0127	0127
0FDB	0FDB	0FDB
:	:	:
:	:	:
0E99	0E99	0E99
:	:	:
:	:	:
0E99	0E99	0E99
0E9A	0E9A	0E9A
04CF*	04DF**	04DF**
0B79	0FB6^	0FB7^^

* Microstate 04AF is the normal entry point.

** Microstate 04BF is the entry point for both violations.

^ Entry point for SPTE invalid

^^ Entry point for SPTE access violation

Figure 5-4. Microflow study for violations in system space.

This series of microsequences analyzes various violations in length of either system or P0 space.

Normal Flow	P0 Length	System Length
0127	0127	0127
0FDB	0FDB	0FDB
:	:	:
:	:	:
0B60	0B60	0B60
023F	023F	023F
045F	045F	045F
0E7D	0E7D	0E7D
0E80	0E80	0E80
0FA7*	0FA3**	0FA7
0E81	0B62	0E81
0E84		0E84
:		:
:		:
0E94		0E94
0E96		0E96
0FAF^		0FAB^^
0E98		0B7A

`*,^ - Normal entry points for microsequence

** - Entry point for P0 length violation

^^ - Entry point for System length violation

Figure 5-5. Microflow study for length violations.

Below is the listing of all the microstates that are needed by VAX-11/780 to perform the program under study for this course without memory management.

0127
0FDB
03DF
00AB
0062
0062
0062
0062
0062
0062
0062
0062
0009
0094
0224
0062
0009
0094
0224
0224
0224
0224
0224
0224
0224
0224
0224
0224
0224
0224
0224
0224
0224
0062
0004
02C4
0062
008F
039F
00FF

Figure 5-6. Normal flow with memory management off.

SUPPLEMENT B – V-BUS LISTINGS

CPT0 UPC=0062

>>>E/VB 0/N:62

```

VB 00000000 E062E02D4515191C7CF1E10062
VB 00000001 F7C08179C2F9FE38
VB 00000002 DA3F470E10080013DFFFC34E00
VB 00000003 809F02E6CB7F0000FFB6C937
VB 00000004 71E020AAAB4000006555FC40E078
VB 00000005 0000FFFF93E000B00000FC0003E9C1E7
VB 00000006 FFFFFFFFFFFFFFFF
VB 00000007 FFFFFFFFFFFFFFFF
VB 00000000 E062E02D4515191C7CF1E10062
VB 00000001 F7C08179C2F9FE38
VB 00000002 DA3F470E10080013DFFFC34E00
VB 00000003 809F02E6CB7F0000FFB6C937
VB ^00000004C 71E020AAAB4000006555FC40E078

```

>>>N

CPT1

>>>E/VB 0/N:6

```

VB 00000000 E062E02D4515191C7CF1610062
VB 00000001 F7C08179C2F9FE3F
VB 00000002 DA3F470E10080013DFFFC34E00
VB 00000003 809F02A6CB7F0000FFB6C93D
VB 00000004 71E020AAAA8000006555FC80E078
VB 00000005 0000FFFF93E000B00000FC0003E9C1E7
VB 00000006 FFFFFFFFFFFFFFFF

```

>>>N

CPT2

>>>E/VB 0/N:6

```

VB 00000000 E062E02D4515191C7CF1610062
VB 00000001 F7C08179C2F9FE38
VB 00000002 DA3F470E10080013DFFFC34E00
VB 00000003 809F02A6CB7F0000FFB6C93D
VB 00000004 F1E020AAAA4800006555FC60E078
VB 00000005 0000FFFF93E000B00000FC0003E9C1E7
VB 00000006 FFFFFFFFFFFFFFFF

```

>>>N

CPT3 APC=0000

>>>E/VB 0/N:6

```

VB 00000000 E062E02D4515191C7CF1E10062
VB 00000001 F7C08179C2F9FE38
VB 00000002 DA3F470E10080013DFFFC34E00
VB 00000003 809F02A6CB7F0000FFB6C93D
VB 00000004 F1E020AAAA4000006555FC50E078
VB 00000005 0000FFFF93E000B00000FC0003E9C1E7
VB 00000006 FFFFFFFFFFFFFFFF

```

CPT0 UPC=0062

>>>E/VB 0/N:6

```

VB 00000000 E062E02D4515191C7CF1E10062
VB 00000001 F7C08179C2F9FE38
VB 00000002 DA3F470E10080013DFFFC34E00
VB 00000003 809F02E6CB7F0000FFB6C937
VB 00000004 F1E020AAB4000006555FC40E078
VB 00000005 0000FFFF91A00C300000DC0003E3C1E7
VB 00000006 FFFFFFFFFF

```

>>>N

CPT1

>>>E/VB 0/N:6

```

VB 00000000 E062E02D4515191C7CF1610062
VB 00000001 F7C08179C2F9FE3F
VB 00000002 DA3F470E10080013DFFFC34E00
VB 00000003 809F02A6CA7F0000FFB6C93F
VB 00000004 F1E020AAB8000007555DC80E078
VB 00000005 0000FFFF90A006200000DB0003E3C1E7
VB 00000006 FFFFFFFFFF

```

>>>N

CPT2

>>>E/VB 0/N:6

```

VB 00000000 E062E02D4515191C7CF1610062
VB 00000001 F7C08179C2F9FE38
VB 00000002 DA3F470E10080013DFFFC34E00
VB 00000003 809F02A6CA7F0000FFB6C93F
VB 00000004 F1E020AAB4800007555DC68E078
VB 00000005 0000FFFF90A006200000DB0003E3C1E7
VB 00000006 FFFFFFFFFF

```

>>>N

CPT3 APC=0000

>>>E/VB 0/N:6

```

VB 00000000 E062E02D4515191C7CF1E10062
VB 00000001 F7C08179C2F9FE38
VB 00000002 DA3F470E10080013DFFFC34E00
VB 00000003 809F02A6CA7F0000FFB6C93F
VB 00000004 F5E020AAB4000007555DE58E078
VB 00000005 0000FFFF90E006200000FB0003E3C1E7
VB 00000006 FFFFFFFFFF

```

CPT0 UPC=0062

>>>E/VB 0/N:6

```

VB 00000000 E062E02D4515191C7CF1E10062
VB 00000001 F7C08179C2F9FE38
VB 00000002 DA3F470E10080013DFFFC34E00
VB 00000003 809F02E6CB7F0000FFB6C937
VB 00000004 F1E020AAAB4000006555FC40E078
VB 00000005 0000FFFF91E000300000FC0003E9C1E7
VB 00000006 FFFFFFFFFFFFFFFF

```

>>>N

CPT1

>>>E/VB 0/N:6

```

VB 00000000 E062E02D4515191C7CF1610062
VB 00000001 F7C08179C2F9FE3F
VB 00000002 DA3F470E10080013DFFFC34E00
VB 00000003 809F02A6CB7F0000FFB6C93D
VB 00000004 F1E020AAAA8000006555FC80E078
VB 00000005 0000FFFF91E00C300000FC0003E3C1E7
VB 00000006 FFFFFFFFFFFFFFFF

```

>>>N

CPT2

>>>E/VB 0/N:6

```

VB 00000000 E062E02D4515191C7CF1610062
VB 00000001 F7C08179C2F9FE38
VB 00000002 DA3F470E10080013DFFFC34E00
VB 00000003 809F02A6CB7F0000FFB6C93D
VB 00000004 F1E020AAAA4800006555FC60E078
VB 00000005 0000FFFF91E00C300000FC0003E3C1E7
VB 00000006 FFFFFFFFFFFFFFFF

```

>>>N

CPT3 APC=0000

>>>E/VB 0/N:6

```

VB 00000000 E062E02D4515191C7CF1E10062
VB 00000001 F7C08179C2F9FE38
VB 00000002 DA3F470E10080013DFFFC34E00
VB 00000003 809F02A6CB7F0000FFB6C93D
VB 00000004 F1E020AAAA4000006555FC50E078
VB 00000005 0000FFFF91E00C300000FC0003E3C1E7
VB 00000006 FFFFFFFFFFFFFFFF

```

CPT0 UPC=0062

>>>E/VB 0/N:6

```

VB 00000000 E062E02D4515191C7CF1E10062
VB 00000001 F7C08179C2F9FE38
VB 00000002 DA3F470E10080013DFFFC34E00
VB 00000003 809F02E6CB7F0000FFB6C937
VB 00000004 F1E020AAB4000006555FC40E078
VB 00000005 0000FFFF91E000B00000FC0003E9C1E7
VB 00000006 FFFFFFFFFFFFFFFF

```

>>>N

CPT1

>>>E/VB 0/N:6

```

VB 00000000 E062E02D4515191C7CF1610062
VB 00000001 F7C08179C2F9FE3F
VB 00000002 DA3F470E10080013DFFFC34E00
VB 00000003 809F02A6CB7F0000FFB6C93D
VB 00000004 F1E020AAA8000006555FC80E078
VB 00000005 0000FFFF91E000300000FC0003E9C1E7
VB 00000006 FFFFFFFFFFFFFFFF

```

>>>N

CPT2

>>>E/VB 0/N:6

```

VB 00000000 E062E02D4515191C7CF1610062
VB 00000001 F7C08179C2F9FE38
VB 00000002 DA3F470E10080013DFFFC34E00
VB 00000003 809F02A6CB7F0000FFB6C93D
VB 00000004 F1E020AAA4800006555FC60E078
VB 00000005 0000FFFF91E000300000FC0003E9C1E7
VB 00000006 FFFFFFFFFFFFFFFF

```

>>>N

CPT3 APC=0000

>>>E/VB 0/N:6

```

VB 00000000 E062E02D4515191C7CF1E10062
VB 00000001 F7C08179C2F9FE38
VB 00000002 DA3F470E10080013DFFFC34E00
VB 00000003 809F02A6CB7F0000FFB6C93D
VB 00000004 F1E020AAA4000006555FC50E078
VB 00000005 0000FFFF91E000300000FC0003E9C1E7
VB 00000006 FFFFFFFFFFFFFFFF

```

CPT0 UPC=0062

>>>E/VB 0/N:6

```

VB 00000000 E062E02D4515191C7CF1E10062
VB 00000001 F7C08179C2F9FE38
VB 00000002 DA3F470E14080013DEF7C34E90
VB 00000003 809F32F60EDF8109FFB6C937
VB 00000004 F1E020AAAB4000007555DC48E078
VB 00000005 0000FFFF91A006300000DC0003E3C1C7
VB 00000006 FFFFFFFFFFFFFFFF

```

>>>N

CPT1

>>>E/VB 0/N:6

```

VB 00000000 E009E02D4515191D7CF1610062
VB 00000001 F7C08179C6FFFE3F
VB 00000002 9A3F470E16080013DEF7C34E90
VB 00000003 809F32B60FDF8109FFB6C93F
VB 00000004 F1F020AAAB8000006555FC88E078
VB 00000005 0000FFFF90A000900000C00003E9C1C7
VB 00000006 FFFFFFFFFFFFFFFF

```

>>>N

CPT2

>>>E/VB 0/N:6

```

VB 00000000 E009E02D4515191D7CF1610062
VB 00000001 F7C08179C6FFFE38
VB 00000002 9A3F470E16080013DEF7C34E90
VB 00000003 809F32B60FDF8109FFB6C93F
VB 00000004 F1F020AAAB4800006555FC68E078
VB 00000005 0000FFFF90A000900000C00003E9C1C7
VB 00000006 FFFFFFFFFFFFFFFF

```

>>>N

CPT3 APC=0000

>>>E/VB 0/N:6

```

VB 00000000 E009E02D4515191D7CF1E10062
VB 00000001 F7C49179C6FFFE38
VB 00000002 9A3F470E16080013DEF7C31E90
VB 00000003 809FB2B60FDF8109FFB6C93F
VB 00000004 F5F020AAAB4000006555FE58E078
VB 00000005 0000FFFF90E000900000E00003E9C1C7
VB 00000006 FFFFFFFFFFFFFFFF

```

CPT0 UPC=0009

>>>E/VB 0/N:6

```

VB 00000000 E009E02D4E15991D78F1E10009
VB 00000001 F7C4917986FFFE18
VB 00000002 D91F471614080013F7EFD91E90
VB 00000003 9E9FB2F64B504A09FFB6C937
VB 00000004 F1F020AAAB4000006555FC4AE078
VB 00000005 0000FFFF91E000F00000FC0003E9C1E7
VB 00000006 FFFFFFFFFFFFFFFF

```

>>>N

CPT1

>>>E/VB 0/N:6

```

VB 00000000 E094E02D4E15991D78F1E10009
VB 00000001 F7C4917986FFFE1F
VB 00000002 D91F471616080013F7EFD91E90
VB 00000003 9E9FB2B64B504A09FFB6C93D
VB 00000004 F1F020AAAA8000006555FC8AE078
VB 00000005 0000FFFF91E000D00000E40003F9C1E7
VB 00000006 FFFFFFFFFFFFFFFF

```

>>>N

CPT2

>>>E/VB 0/N:6

```

VB 00000000 E094E02D4E15991D78F1E10009
VB 00000001 F7C4917986FFFE18
VB 00000002 D91F471616080013F7EFD91E90
VB 00000003 9E8FB2B64A504A09FFB6C93D
VB 00000004 71F020AAAA4800006555FC62E078
VB 00000005 0000FFFF93E000D00000E40003F9C1E7
VB 00000006 FFFFFFFFFFFFFFFF

```

>>>N

CPT3 APC=0000

>>>E/VB 0/N:6

```

VB 00000000 E094E02D4E15991D78F1E10009
VB 00000001 F7C4917986FFFE18
VB 00000002 D91F471616080013F7EFD91E90
VB 00000003 9E8FB2B64A504A09FFB6C93D
VB 00000004 71F020AAAA4000006555FC52E078
VB 00000005 0000FFFF93E000D05281E5520FF9C1E7
VB 00000006 FFFFFFFFFFFFFFFF

```

SUPPLEMENT C – MICROSTATES FOR PROGRAM UNDER STUDY

This series of microstates is necessary to perform the following program (this is a partial listing):

Virtual Address	Macrocode
400	MOVB (R1)+, R2
403	MOVB (R3)+, R4
406	ADDB3 R2, R4, R5
410	Halt

This program was loaded from the floppy using a console command file. All base registers and other necessary data for this image to run within the context of the process have also been loaded. Once the hardware PCB has been loaded by the load context macro instruction by >>>D PC 80000050
>>>C

the program will run to a halt state. Then the operator performed the following.

```
>>>I      /invalidates cache and translation buffer
>>>D/ID 12 1 /turns on memory management which Init clears
>>>D/ID 21 127 / Sets up microbreak register
>>>S SOMM   /stop on micromatch
>>>D PC 400 /sets up starting address in PC
>>>C      /continue
```

The program will start to run but will stop in ROM state 127. Single step from that point creates

Microstate	Comments (highlights only)
0127	stops here for Console Continue or Start
0FDB	test for compatibility mode
03DF	clears console mode
00AB	increment the PC (PC+1)
0062	IRD state (instruction decode)
0060	IB (instruction buffer) miss, 1st push on microstack this address is an entry point for IB miss
0E64	get VIBA
0E66	set up for translation
0B60	go get PTE, 2nd push on microstack
023F	extract the VPN
045F	get P0BL
0E7D	test for length
0E80	passed length test, VPN shifted left 2, get P0BR
0FA7	calculate PxPTE
0E81	set up test for PxPTE

```

0E84      test if PxPTE is in the TB (translation buffer)
025F      PxPTE not in TB
0E85      extract VPN from PTE.VA
0B78      SLR to Q, 3rd push on the microstack
0E94      set up for length violation
0E96      test for length violation
0FAF      passed length test, calculate SPTE physical address
0E98      fetch SPT into D register
0E99      mask out the MBZ field of SPTE
0E9A      test valid, not no access and MBZ
04CF      passed test, load SPTE into TB, return
0B79      fetch PxPTE
0E8A      mask out MBZ bits
0E8C      test PTE for valid, not no access and MBZ
04AF      passed test, load PTE into TB, return
0B61      restart IB, load IPA, restores some entry registers
0E68      restores PC to entry value
0E69      return
0062      IRD
0009      A fork entry for (R)
0094      allows for TB miss (read virtual)
0224      this pc will be bused on microstack due to TB miss
0105      TB miss microtrap (entry point for data stream)
0E42      get ready for translation
0E42      set up for translation
0B42      call to calculation for TB
023F      notice that the microcode has been here before
045F      and for the next five. Same as above
0E7D
0E80
0FA7
0E81

0E84      test if PxPTE is in TB
025C      SPTE for PxPTE VA is in TB
0E8A      mask out MBZ bits
0E8C      test for valid, not no access and MBZ
04AF      passed test, load into TB
0B49      check for PTE M bit
0D6C      setting of M bit not required
0B48      return from test
0E48      restores register, retries memory reference trap
0D1C      read memory reference, return
0224      B fork, write register
0062      IRD

```


SUPPLEMENT D – DISCUSSION OF MICROSTATE 62 – IRD (INSTRUCTION REGISTER DECODE)

This state is the primary instruction register decode state. It is one of many states that contain the same code, but this one is the one most commonly used throughout the microcode. Let's examine this state by first looking at the nontransfer function macros that define it.

According to the definition of this state given in the middle column, this state performs one function: the nontransfer function IRD. This macro will expand to the following by definition:

IRD0, CLK.UBCC, IRD1, SUB/SPEC, J/A.FORK

These macros will be defined eventually which will allow the entire processor to be set up for the next instruction to be fully optimized. Each register within data paths will be loaded with some portion of the operands, according to specifiers 1 and 2. These will enable the next instruction to execute in as few states as possible. No matter what the modes of the instructions are, they will be easily reconciled by this state. Also, the registers that are set up in this state will enable the next instruction to be implemented quickly, no matter what type it is. Integer, logical, and floating-point instruction types are the three types that are considered most often since most programs will usually use these types.

To enable us to clearly see how this optimization will take place, let's break down this microstate until we have each of the fields within the microword defined.

First, let us break down the macros that define the IRD macro itself.

IRD0 definition

LA_R(SP2)&LB_R(SP1). These macros will load the general purpose latches with the appropriate register contents according to the first two operands. Note that if there are not two operands, then latch A will be loaded with junk.

D&VA_LB. This macro will take the contents of latch B and load it into both the D-register and the virtual address register. This is being done primarily for the following two cases.

If the instruction's first mode is register mode, this means that we have just loaded the first operand into the the D-register so that it can be accessed and manipulated quickly. For example, if the instruction was an 'ADDL2 R4, R3', then the contents of R4 has just been loaded into the D-register, and in one state, we can send it through the ALU and add it to the contents of latch A which contains the contents of R3.

If the mode of the first operand is any mode other than short literal or index, we will have to take the contents of the register and use it as an address. This means we will have to make a memory reference using the contents of the register. The contents of the first register is now sitting in the VA register, has already been to the translation buffer in the next state and can perform the read.

SC ALU(EXP). This macro will take the contents of the ALU bits <14:07> and load them into the SC register. This is for floating-point operations where the fraction, exponent and sign bits must be handled in separate areas of the data paths. The exponent of the first operand, then, has been loaded into the SC and is ready for processing in the exponent section of the data paths since the SC has direct access to it through the EAMX.

FE_LA(EXP). Since the ALU is performing a function of B to load the D and VA registers, and we can perform no arithmetic operations since: (1) we have just latched the GPRs in this state, and (2) we do not yet know what the instruction is, it is impossible to perform any other function. Therefore, since latch A has been loaded with the contents of the register specified by the second operand of the instruction, we can still perform some type of logical operation on the data. We will load the exponent bits of this operand into the FE register to set up for a register-to-register floating-point operation. This can be done quite simply by sending the exponent through the EBMX, into the EALU (note that a function of B is a logical function), through the EMUX and into the FE register. Now we can perform an add, subtract, etc. on both of the exponents since both have direct access in one state to the EALU.

SS_ALU15. If you remember, we stated earlier that each of a floating-point number needs to be operated on in separate sections of the data paths. The sign section is not shown on the block diagram since it is very small, but it does exist on the control board. We have just taken the output of bit <15> of the ALU and loaded it into the "sign of the source" register. Since the total output of the ALU is the contents of the first operand register, and the first operand is the source, this is very appropriate.

CLK.UBCC definition

This macro allows the microbranch condition codes to be clocked at this time. This will allow us to branch off the type of instruction we are performing in the next state.

IRD1 definition

MSC/IRD. This is a field definition since the statement contains a /. This means that the miscellaneous field of the microword will receive an IRD code. This is very important since it performs two major functions. First, it initializes the contents of the RLOG stack pointer to zero, which indicates that there have been no autoincrement or autodecrement operations performed on any of the registers in this instruction. This is good since we do not know yet what the modes of the operands are, or even if there are any operands to be processed (one byte instructions). Second, it allows the ICLD service bit codes to be strobed so we can branch in the case of an instruction buffer stall, arithmetic trap, interrupt, etc.

QK/ID. This is also a field definition and it means that the Q-register is to be loaded from the ID bus. This is important where the instruction being executed contains some type of literal data or displacement data. In either of these cases, the data needs to be placed into data paths so that it can be operated on. Let's examine both cases more closely:

Literal data - Whether the data is short literal, or long literal, by putting the data into the Q-register we can now use that data (which is the first operand) in the next state. If the instruction was an 'Addl2 #20, R3', we can simply add the contents of Q to the contents of latch A (which now contains the contents of the register specified by the second specifier; in this case, R3) and put the answer right back into R3. If the data is quad literal, then we have just freed the necessary bytes within the IB to allow more prefetch of data to get the remainder.

Displacement data - In this case we have obtained the necessary data to be added to some register to reload the VA with its contents. This can be done in one state by simply taking the contents of latch B, adding it to the Q-register, and placing the sum into the VA. We have just set up the address of the first operand and are ready to make the appropriate memory reference.

MCT/ALLOW.IB.READ. This field allows the IB to make a memory reference to obtain more I-Stream data if necessary. This will allow the prefetch of the next instruction or more of the present instruction to be operated on. This can be done at this time since it is impossible for data paths to be performing any kind of memory reference. Also, this is a very necessary field value since it must make a memory reference if the IB does not have enough data to decode the present instruction.

IBC/CLR.1-5.COND. This field value allows the IB to clear those bytes that it no longer needs and to shift down the contents of the next bytes for subsequent decoding. The bytes being discarded will need to be accounted for in both the VIBA and the PC. The first is hardware controlled and the second is microcode controlled (see the next microfield discussed for the PC control definition for this state).

PCK/PC+N. This field value allows the PC to be updated according to the value of N, where N is determined by the instruction buffer. This is true since the IB is the only part of the CPU that knows just how many bytes of data it is actually discarding.

SUB/SPEC definition

This is a microfield definition that allows the decode logic on the IRC (M8224 - Instruction Register Decode) to tell the microsequencer where in the microcode it should direct itself based on the decode of both the instruction and the mode(s) of the operand(s). This is accomplished through allowing the IRC board to output the low 8 bits of the next micro PC; that is, micro-PC <07:00>. The decision made by the IRC board is based on a multiplexer that is on page IRCE in the upper right-hand corner of the page. As you can see, this decision is based upon many gating factors. These factors not only include the instruction and addressing modes, but also include some service bit codes that will cause a special branch if there has been an interrupt, exception, etc.

J/A.FORK definition

This field value sets the contents of the JMP field to a setting of zero. This is done because the base state of A-FORK must be zero in order to allow the instruction buffer to tell us where in A-FORK we must go next. Note that because of the SUB/SPEC micro, we have the possibility of entering into A-FORK at 255 different places. This is because the IB determines the low 8 bits of the next micro-PC. Also note that we must have these bits as zeros since the IB micro PC bits will be logically ORed with those in the JMP field, and anything other than zeros may cause us to go to the wrong state.

BLANK

SUPPLEMENT E - COMMAND FILE FOR PROGRAM UNDER STUDY

```

INIT
UNJAM
DEPOSIT PSL 04000000      ! Clear out PSL (Kernel mode, IPL=>0,
!                          ! ISP=>1)
! The following deposit is to clear memory
! DEPOSIT 0 0/NEXT:1000
!
! Set up the Interrupt/Exception Vectors to HALT the machine.
! DEPOSIT 0 3/NEXT:80      ! One full page
!
! Set up the interrupt/exception vectors
!
DEPOSIT 20 80000100
DEPOSIT 24 80000000
!
! Deposit the program into P0 space
!
DEPOSIT 200 90528190      ! MOVB (R1)+, R2
DEPOSIT + 52815483        ! MOVB (R3)+, R4
DEPOSIT + 00005554        ! ADDB3 R2, R4, R5
!                          ! HALT
DEPOSIT 300 90528190      !
DEPOSIT + 52815483        ! Do it all over again in different
!                          ! addresses
DEPOSIT + 00005554        !
DEPOSIT 400 0000000A      ! First P0 operand
!
! Deposit the pagefault routine
!
DEPOSIT 800 885857D0      ! MOVL R7, R8
DEPOSIT + 03AA808F        ! BISB #^X80, 3(R10)
DEPOSIT + 025E08C0        ! CLEAN UP STACK
!                          ! REI
!
! Deposit the operating system code in for scheduler
!
DEPOSIT 850 00000206      ! LDPCTX
!                          ! REI
!
! Deposit the operating system code for context switching and
! access violation
! correction routine (exception handler)
!
DEPOSIT 900 D05E08C0      ! CLEAN STACK UP
DEPOSIT + 50DA5957        ! MOVL R7, R9
DEPOSIT + 8F880710        ! MTPR R0, #PR$_PCBB
DEPOSIT + DA03AAA0        ! SVPCTX
DEPOSIT + 02061050        ! BISB #^XA0, 3(R10)

```

```

! MTPR R0, #PR$_PCBB
! LDPCTX
! REI

!
! Deposit the P0PT (P0 Page Table) into memory
!
DEPOSIT A00 00000000
DEPOSIT + A0000002
DEPOSIT + A0000001
DEPOSIT + A0000006
!
! Deposit the second P0 space operand
!
DEPOSIT C00 0000000B
!
! Deposit the P1PT (P1 Page Table) into memory
!
DEPOSIT E00 A000000C
DEPOSIT + 00000000
DEPOSIT + F000000F
DEPOSIT + E800000D
DEPOSIT + E000000E
DEPOSIT + 00000000
!
!
! Deposit clean up routine for Save Context instruction to
! simulate VMS
!
DEPOSIT 1150 80000A00
DEPOSIT + 04000004
DEPOSIT + 7F800C18
DEPOSIT + 001FFFFA
!
! Deposit the new hardware PCB (Process Control Block) into memory
!
DEPOSIT 1200 7FFFFFFA00      ! KSP
DEPOSIT + 7FFFFC00           ! ESP
DEPOSIT + 7FFFFE00           ! SSP
DEPOSIT + 7FFFF600           ! USP
DEPOSIT + 00001100           ! R0
DEPOSIT + 00000200           ! R1
DEPOSIT + 00000000           ! R2
DEPOSIT + 00000600           ! R3
DEPOSIT + 00000000           ! R4
DEPOSIT + 00000000           ! R5
DEPOSIT + 00001200           ! R6
DEPOSIT + 12345678           ! R7
DEPOSIT + 00000000           ! R8
DEPOSIT + 00000000           ! R9
DEPOSIT + 80000A08           ! R10
DEPOSIT + 00000000           ! R11

```



```

DEPOSIT + 00000000      ! R12 (AP)
DEPOSIT + 00000000      ! R13 (FP)
DEPOSIT + 00000400      ! R15 (PC)
DEPOSIT + 00000000      ! PSL (Kernel mode, IPL=>0, ISP=>0)
DEPOSIT + 80000A00      ! P0BR
DEPOSIT + 04000004      ! ASTLVL, P0LR
DEPOSIT + 7F800C18      ! P1BR
DEPOSIT + 001FFFFFA      ! P1LR
!
! Deposit the SPT (System Page Table) into memory
!
DEPOSIT 1400 F0000004
DEPOSIT + F000000B
DEPOSIT + F0000000
DEPOSIT + F0000008
DEPOSIT + F0000009
DEPOSIT + F0000005
DEPOSIT + F0000007
DEPOSIT + F000000A
!
! Fix up all of the IPR's (Internal Processor Registers) that we
! need to make
! the machine work like it's supposed to.
! SET DEFAULT INTERNAL
DEPOSIT 4 80000400      ! ISP
DEPOSIT C 00001400      ! SBR
DEPOSIT D 00000008      ! SLR
DEPOSIT 10 00001200     ! PCBB
DEPOSIT 11 00000000     ! SCBB
DEPOSIT 38 00000001     ! MME - Memory Management is ON!!!!
DEPOSIT 39 00000000     ! TBIA
SET DEFAULT PHYSICAL
!
DEPOSIT/IDBUS 12 00000001 ! Make sure Memory Management is on
!
! Clear out the first 9 GPR's (General Purpose Registers)
!
DEPOSIT R0/NEXT:8 00000000
!
! Set up the current stack pointer - Kernel Mode on Interrupt
! Stack
!
DEPOSIT SP 80000400
!
! The "SAW mini-VMS operating system has been initialized!!!
!
```

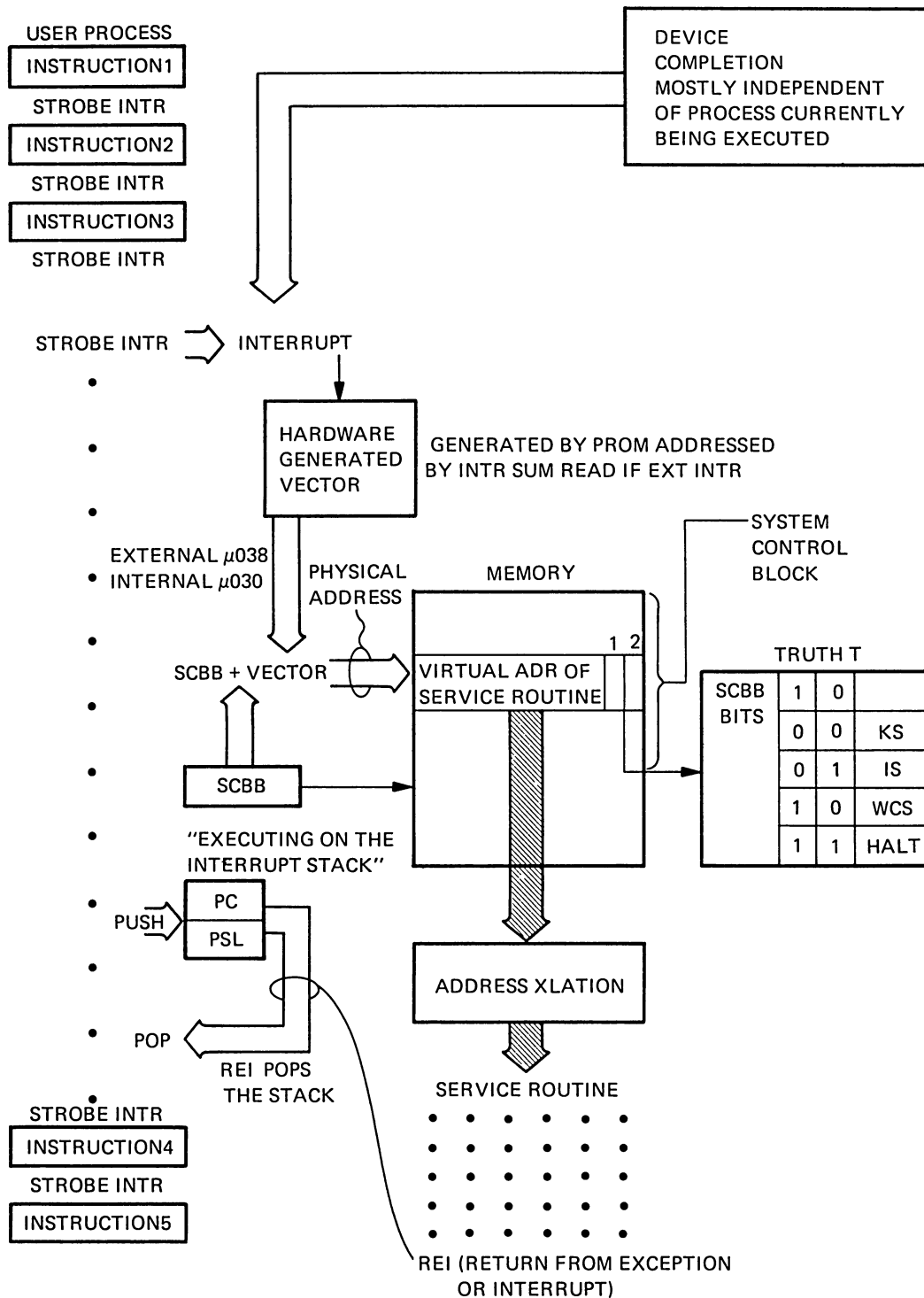
BLANK

SUPPLEMENT F - VAX-11/780 JARGON

1. The operating system (VMS) spends most of its time executing in user mode in the context of one process or another. When user software needs the services of the operating system, whether for acquisition of a resource, for I/O processing, or for information, it calls those services.

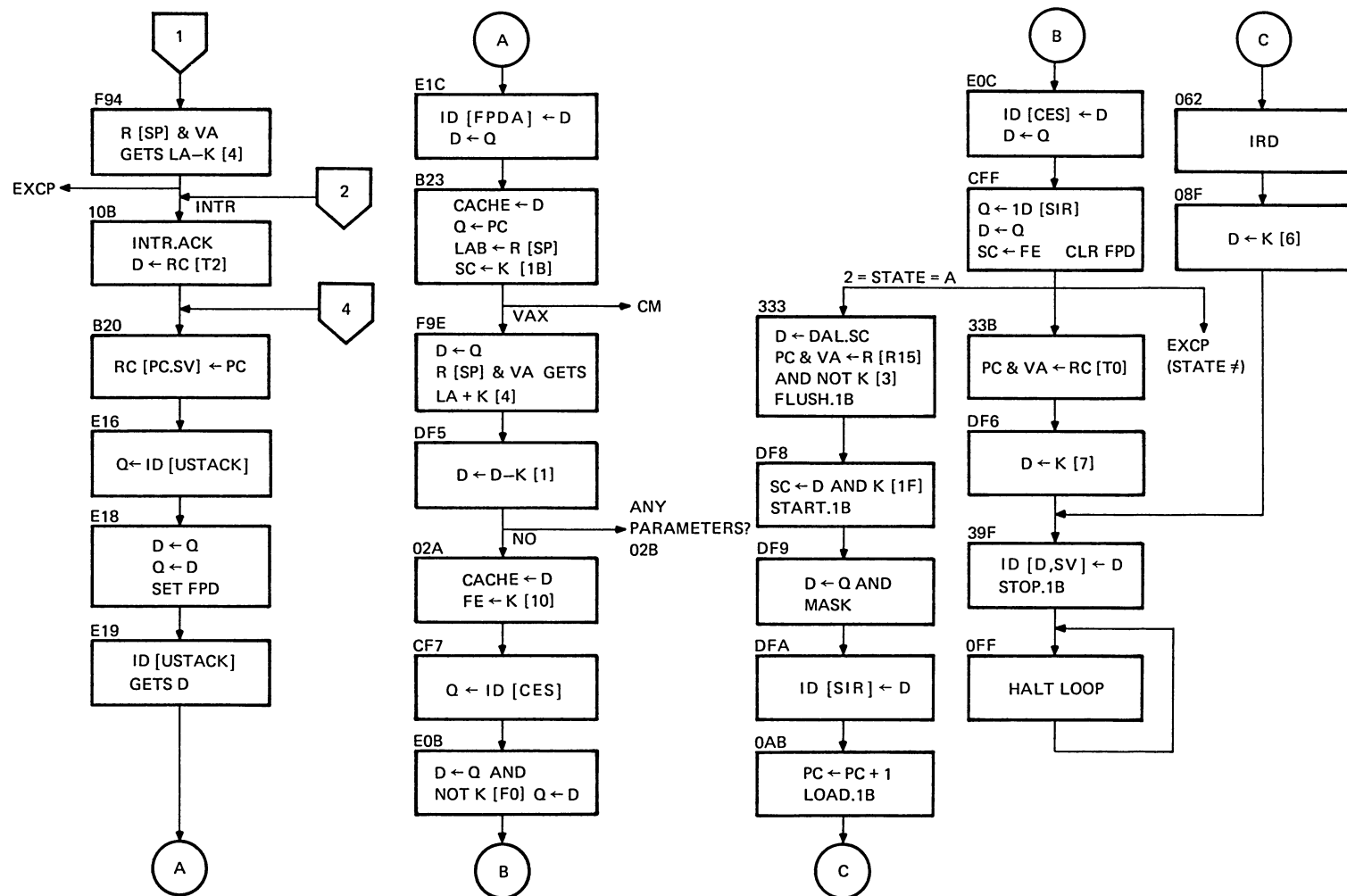
The processor executes those services in the same or one of the more privileged access modes within the context of that process. That is, all four access modes exist within the same virtual address space. Each access mode has its own stack in the control region of per process space (P1 space); therefore, each process has four (4) stacks, one for each access mode.

2. At any one time, the processor is executing code in the context of a particular process, or it is executing in the system-wide interrupt service context (executing on the interrupt stack).
3. An image is an executable program. It is created by translating source language modules into object modules and linking the object modules together. This is achieved by the MARS assembler and linker.
4. The environment in which an image executes is its context. The complete context of an image not only includes the state of its execution at any time (known as its hardware context), but it also includes the definition of its resource allocation privileges and quotas such as device ownership, file access and maximum memory allocation. These privileges and quotas are given to the user who runs the image.
5. An image context, including the address space used by an image, is called a process. The operating system schedules processes, which provide a context in which an image executes.



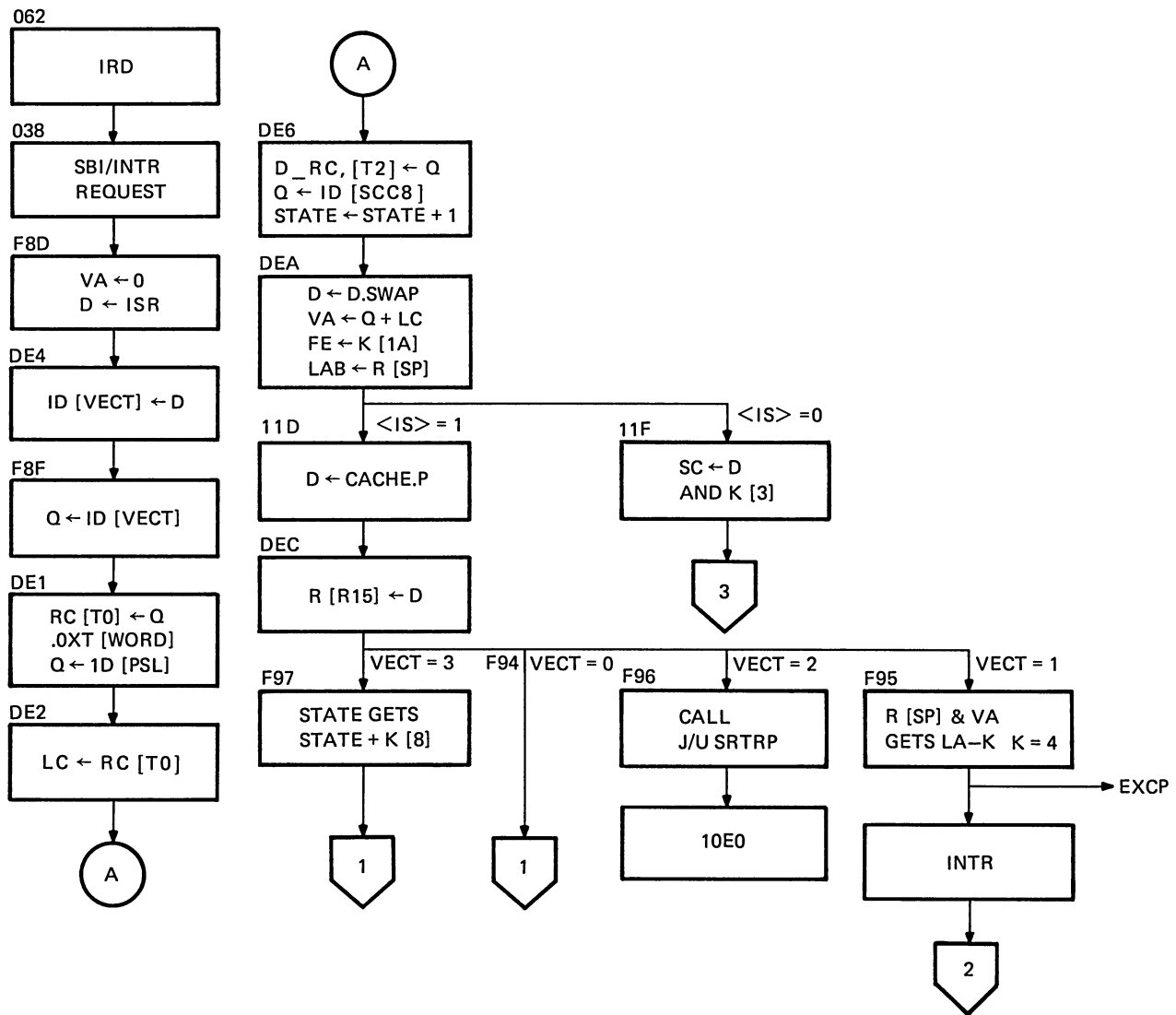
TK-4348

Figure 5-7. Interrupt Overview



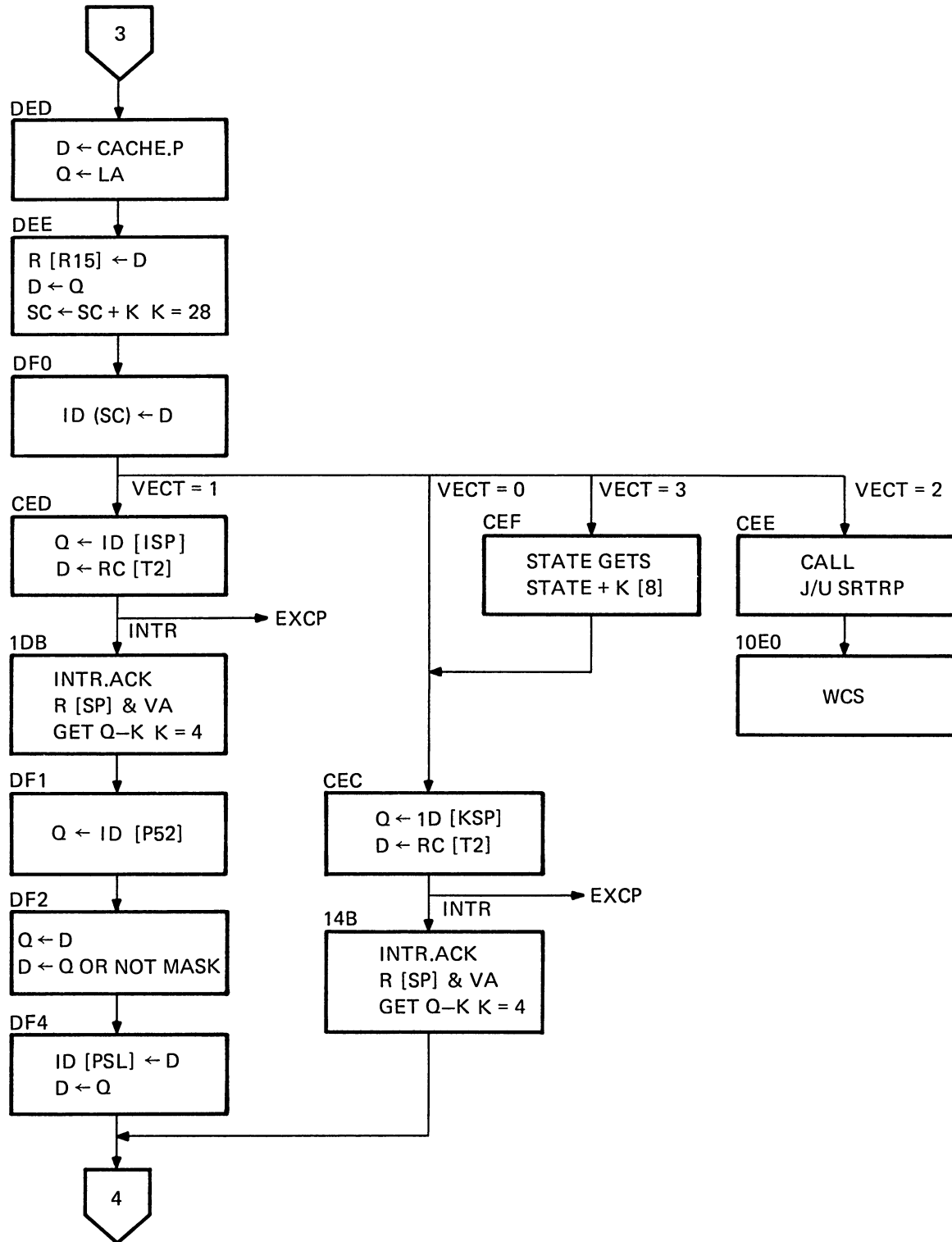
TK-1255

Figure 5-8. External Interrupts, Kernel Mode, and Memory Management Off



TK-1247

Figure 5-9. External Interrupts, Kernel Mode and Memory Management Off (Cont)



TK-1248

Figure 5-10. External Interrupts, Kernel Mode, and Memory Management Off (Cont)

BLANK

SUPPLEMENT G – INSTRUCTION REGISTER DECODE WORKSHEETS

1. E8308 (D5) qualified/disqualified

Signal Name	Level
a. IRCF I MODE (1) H	H/L
b. IRCM BUF B1-6 L	H/L
c. IRCE CTX 1 H	H/L
d. IRCE CTX 0 H	H/L
e. IRCM BUF B1-4 H	H/L
f. IRCM BUF B1-5 L	H/L
g. ASSUME IRCB WRITE ABORT L a high.	

2. E9508 (C5) qualified/disqualified

a. IRCM BUF B1-4 H	H/L
b. IRCM BUF B1-6 H	H/L
c. IRCM BUF B1-5 H	H/L
d. IRCM BUF B1-5 L	H/L
e. IRCM BUF B1-7 H	H/L
f. IRCE CTX 3 H	H/L
g. IRCE CTX 2 H	H/L
h. IRCM BUF B1-6 L	H/L
i. IRCM BUF B1-4 L	H/L
j. IRCD PC MODE L	H/L

3. E9608 (B5) qualified/disqualified

a. IRCM BUF B1-6 H	H/L
b. IRCM BUF B1-5 L	H/L
c. IRCM BUF B1-5 H	H/L
d. IRCM BUF B1-7 H	H/L
e. IRCD PC MODE H	H/L
f. IRCE CTX 2 H	H/L
g. IRCE CTX 3 H	H/L
h. IRCM BUF B1-4 L	H/L

4. E14108 (B5) qualified/disqualified

a. IRCM BUF B1-7 L	H/L
b. IRCM BUF B1-4 L	H/L
c. IRCM BUF B1-6 H	H/L
d. IRCM BUF B1-5 H	H/L

5. E14008 (A5) qualified/disqualified

Assume this to be disqualified for this example - prove on your own.

6. E15408 (D2) qualified/disqualified

- a. IRCB REG WRITE L H/L
- b. IRCF R MODE H H/L
- c. IRCC VAX DECODE 06 H H/L

7. E11108 (C2) qualified/disqualified

- a. IRCF R MODE L H/L
- *b. IDPM VAX SL L H/L
- c. IDPA DST R MODE H H/L
- d. IRCF E MODE (0) H H/L
- e. IRCC VAX DECODE 05 H H/L
- f. IRCC IR04 (1) H H/L
- g. IRCC IR03 (0) H H/L
- h. IRCC IR02 (0) H H/L
- i. IRCC IR00 (1) H H/L
- j. IRCC VAX DECODE 04 H H/L

*What does SL stand for?

8. E11008 (B2) qualified/disqualified

- a. IRCE CTX 0 H H/L
- b. IRCC VAX DECODE 05 H H/L
- c. IRCF R MODE L H/L
- d. IDPM VAX SL L H/L
- e. IDPA DST R MODE H H/L
- f. IRCF E MODE (0) H H/L
- g. IRCC VAX DECODE 04 H H/L

If time permits, try an instruction on your own.

1. E8308 (D5) qualified/disqualified

Signal Name	Level
a. IRCF I MODE (1) H	H/L
b. IRCM BUF B1-6 L	H/L
c. IRCE CTX 1 H	H/L
d. IRCE CTX 0 H	H/L
e. IRCM BUF B1-4 H	H/L
f. IRCM BUF B1-5 L	H/L
g. ASSUME IRCB WRITE ABORT L a high.	

2. E9508 (C5) qualified/disqualified

a. IRCM BUF B1-4 H	H/L
b. IRCM BUF B1-6 H	H/L
c. IRCM BUF B1-5 H	H/L
d. IRCM BUF B1-5 L	H/L
e. IRCM BUF B1-7 H	H/L
f. IRCE CTX 3 H	H/L
g. IRCE CTX 2 H	H/L
h. IRCM BUF B1-6 L	H/L
i. IRCM BUF B1-4 L	H/L
j. IRCD PC MODE L	H/L

3. E9608 (B5) qualified/disqualified

a. IRCM BUF B1-6 H	H/L
b. IRCM BUF B1-5 L	H/L
c. IRCM BUF B1-5 H	H/L
d. IRCM BUF B1-7 H	H/L
e. IRCD PC MODE H	H/L
f. IRCE CTX 2 H	H/L
g. IRCE CTX 3 H	H/L
h. IRCM BUF B1-4 L	H/L

4. E14108 (B5) qualified/disqualified

a. IRCM BUF B1-7 L	H/L
b. IRCM BUF B1-4 L	H/L
c. IRCM BUF B1-6 H	H/L
d. IRCM BUF B1-5 H	H/L

5. E14008 (A5) qualified/disqualified

Assume this to be disqualified for this example - prove on your own.

6. E15408 (D2) qualified/disqualified

- a. IRCB REG WRITE L H/L
- b. IRCF R MODE H H/L
- c. IRCC VAX DECODE 06 H H/L

7. E11108 (C2) qualified/disqualified

- a. IRCF R MODE L H/L
- *b. IDPM VAX SL L H/L
- c. IDPA DST R MODE H H/L
- d. IRCF E MODE (0) H H/L
- e. IRCC VAX DECODE 05 H H/L
- f. IRCC IR04 (1) H H/L
- g. IRCC IR03 (0) H H/L
- h. IRCC IR02 (0) H H/L
- i. IRCC IR00 (1) H H/L
- j. IRCC VAX DECODE 04 H H/L

*What does SL stand for?

8. E11008 (B2) qualified/disqualified

- a. IRCE CTX 0 H H/L
- b. IRCC VAX DECODE 05 H H/L
- c. IRCF R MODE L H/L
- d. IDPM VAX SL L H/L
- e. IDPA DST R MODE H H/L
- f. IRCF E MODE (0) H H/L
- g. IRCC VAX DECODE 04 H H/L

If time permits, try an instruction on your own.

SUPPLEMENT H – INSTRUCTION BUFFER TRUTH TABLES

Table 1

S1	S0	Y3	Y2	Y1	Y0
L	L	I3	I2	I1	I0
L	H	I2	I1	I0	I-1
H	L	I1	I0	I-1	I-2
H	H	I0	I-1	I-2	I-3

This table shows how the shifter shifts in the four bytes of data on the MD lines. Notice that with an LL input on the S1 and S0 lines, the output Y0 will set the input of I0.

Related to the logic IDPN [7 1/2, B], you will find the input to I0 [pin 4 E3] reads BUS MD 24 H. This means that bit 24, the first bit of the fourth byte, will now be the output at Y0 [pin 15, E3] IDPN RMD B3-0.

On the other hand, notice that when S1 and S0 are both high, BUS MD 24 H will be shifted to Y3 IDPN RMD B0-0. RMD means rotated memory data, B0-0 means byte 0, position 0.

Do not equate the lows and highs with an actual count such as LL is equal to 0. This will work itself out later.

Tables for IDPH SEL B2-6 S1/S0 H

The truth tables are as follows.

Table 2 SH0

S1	S0	Output	A
L	L	Byte	4
L	H	Byte	2
H	L	Byte	1
H	H	Byte	0

Table 3 SH1

S1	S0	Output	B
L	L	Byte	5
L	H	Byte	3
H	L	Byte	2
H	H	Byte	1

Table 4 SH2 & 3

S1	S0	Output	C	D
L	L	Byte	6	7
L	H	Byte	4	5
H	L	Byte	3	4
H	H	Byte	2	3

Table 5 SH4 & 5

S1	S0	Output	E	F
L	L	Byte	*	*
L	H	Byte	6	7
H	L	Byte	5	6
H	H	Byte	4	5

Table 6 SH6

S1**	S0	Output	H
L	L	Byte	7
L	H	Byte	6
H	L	High impedance	
H	H	High impedance	

* A ground input

** This input is IDPH SEL B2-6 S1 L which is low when IDPH SEL B2-26 S1 H is high.

Tables for Valid Bit Shifters**Table 7 IDPD SHF B0-VAL**

S1	S0	Output
----	----	--------

L	L	If B4 is valid then B0 will be valid
L	H	If B2 is valid then B0 will be valid
H	L	If B1 is valid then B0 will be valid
H	H	If B0 is valid then B0 will be valid

S1 is controlled by IDPH SEL B0-1 S1 H

S0 is controlled by IDPH B0 S0 H

Table 8 IDPD SHF B1-VAL H

S1	S0	Output
----	----	--------

L	L	If B5 then B1
L	H	If B3 then B1
H	L	If B2 then B1
H	H	If B1 then B1

S1 is controlled by IDPH SEL B0-1 S1 H

S0 is controlled by IDPH SEL B1 S0 H

Table 9 IDPE SHF B2-VAL H

S1	S0	Output
----	----	--------

L	L	If B6 then B2
L	H	If B4 then B2
H	L	If B3 then B2
H	H	If B2 then B2

Table 10 IDPE SHF B3-VAL H

S1	S0	Output
----	----	--------

L	L	If B7 then B3
L	H	If B5 then B3
H	L	If B4 then B3
H	H	If B3 then B3

Table 11 IDPH SHF B4-VAL H

S1	S0	Output
L	L	Invalidates B4
L	H	If B6 then B4
H	L	If B5 then B4
H	H	If B4 then B4

Table 12 IDPF SHF B5-VAL H

S1	S0	Output
L	L	Invalidates B5
L	H	If B7 then B5
H	L	If B6 then B5
H	H	If B5 then B5

Table 13 IDPH SHF B6-VAL H

S1	S0	Output
L	L	Invalidates B6
L	H	Invalidates B6
H	L	If B7 then B6
H	H	If B6 then B6

Table 14 IDPH SHF B7-VAL H

S1	S0	Output
L	L	Invalidates B7
L	H	Invalidates B7
H	L	Invalidates B7
H	H	If B7 then B7

S1 for valid shifter bits 2-7 is controlled by IDPH SEL B2-6 S1 H.
 S0 for valid shifter bits 2-7 is controlled by IDPH SEL B2-6 S0 H.
 Bytes are invalidated by shifting in ground.

IDPH B2-6 S1/S0 H, IDPH B1 S0 H, IDPH B0 S0 H and IDPH B0-1 S1 H.
 These signals are driven from three areas:

1. IDB Field 3:0
2. IDPM Delta PC
3. IRCE SAVE H

For the sake of brevity, the following table has been developed,
 and until proven wrong, will be the key of this operation.

Table 15 shows the relationship of two signals:

1. IDPH SEL B2-6 S0 H/L
2. IDPH SEL B2-6 S1 H/L

These two signals will modify the IBA depending upon their levels.

Table 15 IBA

S1H	S0H	S1L	S0L	IBA1(1)H	IBA0(0)H	IBA1(0)H	IBA0(0)H	COUNT	
								IBA	IBA
L	L	H	H	L	L	H	H	0	0
L	H	H	L	L	L	H	H	0	2
H	L	L	H	L	L	H	H	0	1
H	H	L	L	L	L	H	H	0	0
L	L	H	H	L	H	H	L	1	1
L	H	H	L	L	H	H	L	1	3
H	L	L	H	L	H	H	L	1	2
H	H	L	L	L	H	H	L	1	1
L	L	H	H	H	L	L	H	2	2
L	H	H	L	H	L	L	H	2	0
H	L	L	H	H	L	L	H	2	3
H	H	L	L	H	L	L	H	2	2
L	L	H	H	H	H	L	L	3	3
L	H	H	L	H	H	L	L	3	1
H	L	L	H	H	H	L	L	3	0
H	H	L	L	H	H	L	L	3	3

The count shows the initial value of the IBA before and after the select lines have modified the count. The actual IBA counter will only change after each T0.

Another important function to analyze is what effect delta-PC has on the select lines S1/S0 when IBC field equals F and delta-PC makes the only changes that the logic will allow. In other words, delta-PC can only change by 0, 1, 2, 3 and 5.

Table 16

Delta-PC	S1H	S0H
0	H	L
1	H	L
2	H	L
3	L	H
5	L	L

Now you can predict the change of the IBA or any delta-PC when IBC field equals F.

You can also analyze those same select lines for any other value in the IBC's field and IBC's effect on those select lines.

Table 17 IBC

IBC	S1H	S0H	
0	H	H	
1	H	H	
2	H	H	
3*	H	H	*If IBC=3+7+F and IRCE SAVE H is high then S1 H and S0 H are high.
4	L	H	
5	L	H	
7*	H	H	
C	H	L	
D	H	L	
E	L	L	
F*	**	**	** Dependent on delta PC

The select lines for bytes 0 and 1 are changed by the IBC field as follows.

IDPH SEL B0-1 S1 H

This signal is high under the following conditions:

$$IBC=0+1+2+3+5+7+C+D+F$$

This signal is low under the following conditions: IBC=4+E

IDPH SEL B0 S0 H

This signal is high under the following conditions:

$$IBC=0+1+2+3+4+5+7+D+F$$

This signal is low under the following conditions: IBC=C+E

IDPH SEL B1 S0 H

This signal is high under the following conditions:

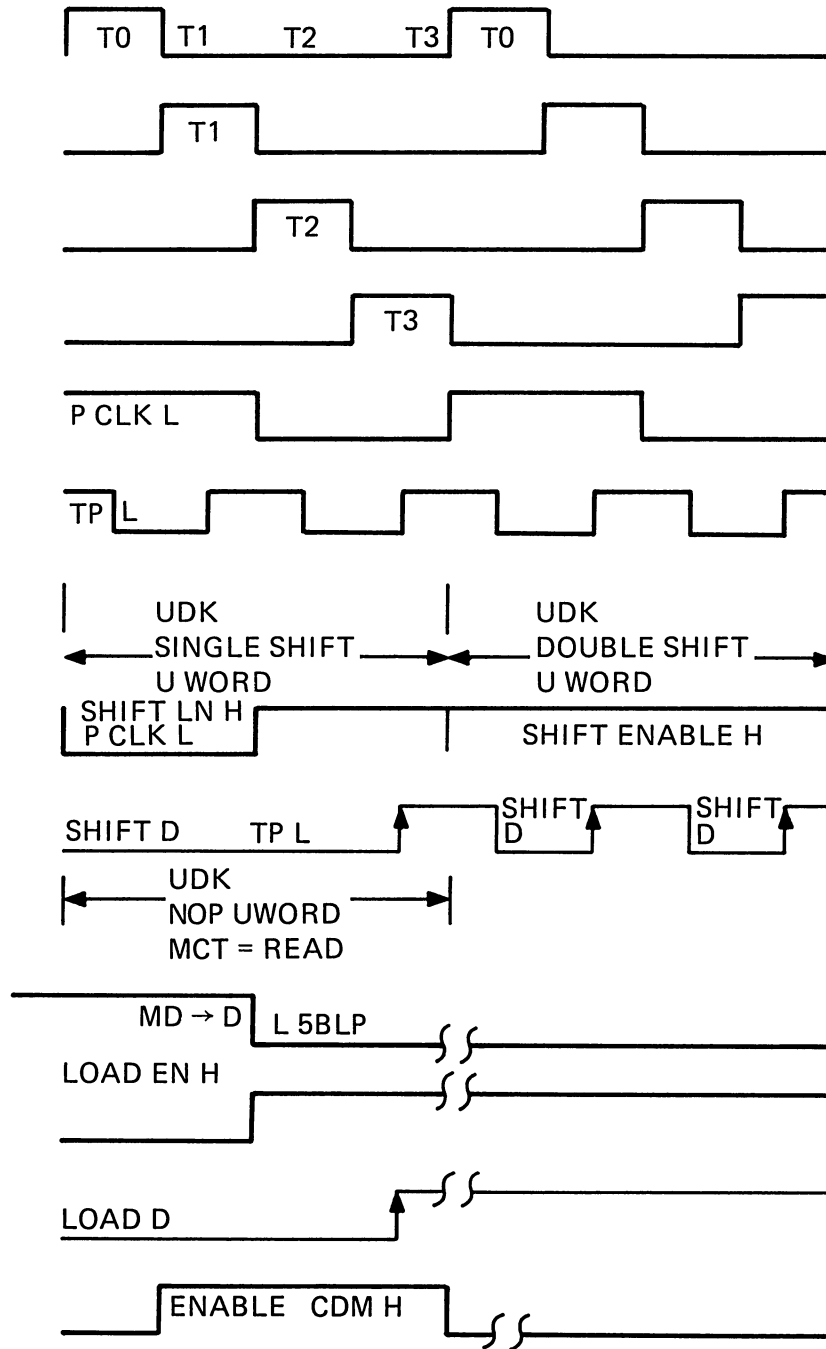
1. Delta PC=5 and IBC=3+7+F OR
2. IBC=3+7+F and IRCE SAVE HOR
3. IBC=0+1+2+3+4+5+7

This signal is low under the following conditions:

All other conditions that were not mentioned for the high.

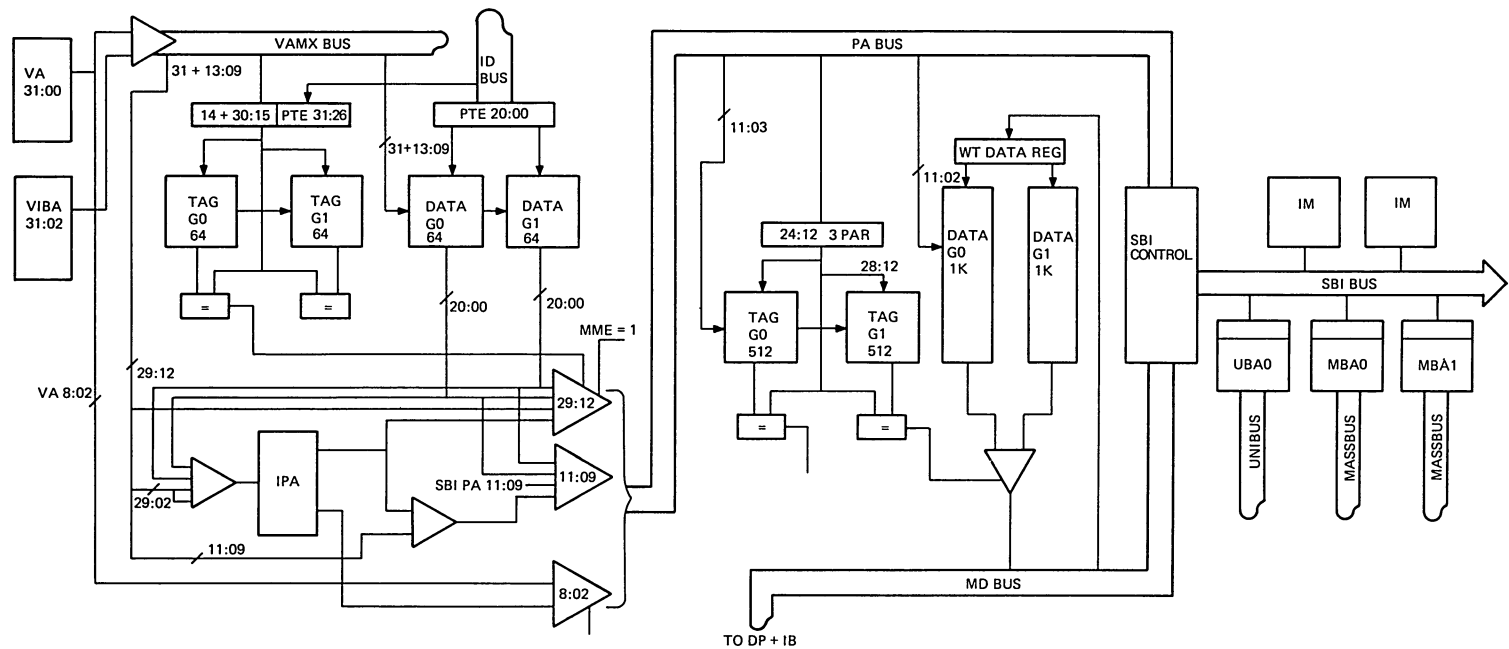
Table 18 Effect of IBA Count on Rotated Memory Data

IBA Count	IDPM IBA1L	IDPM IBA0L	IDPM BIBA1L	IDPM BIBA0L	Y3	Y2	Y1	Y0
0	H	H	L	L	I3	I2	I1	I0
1	H	L	L	H	I2	I1	I0	I-1
2	L	H	H	L	I1	I0	I-1	I-2
3	L	L	H	H	I0	I-1	I-2	I-3



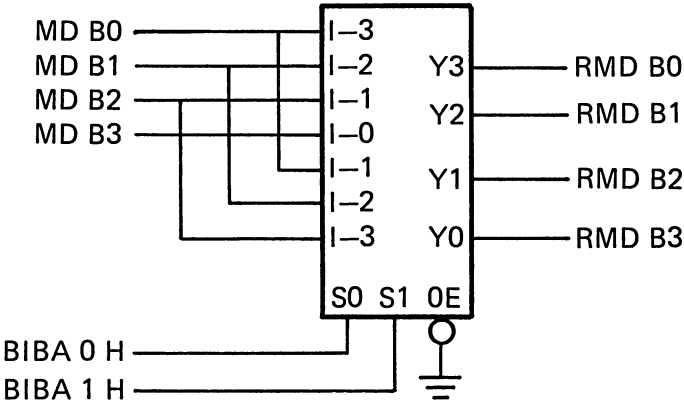
TK-4335

D Register Timing
(Q Register Similar)



TK-4357

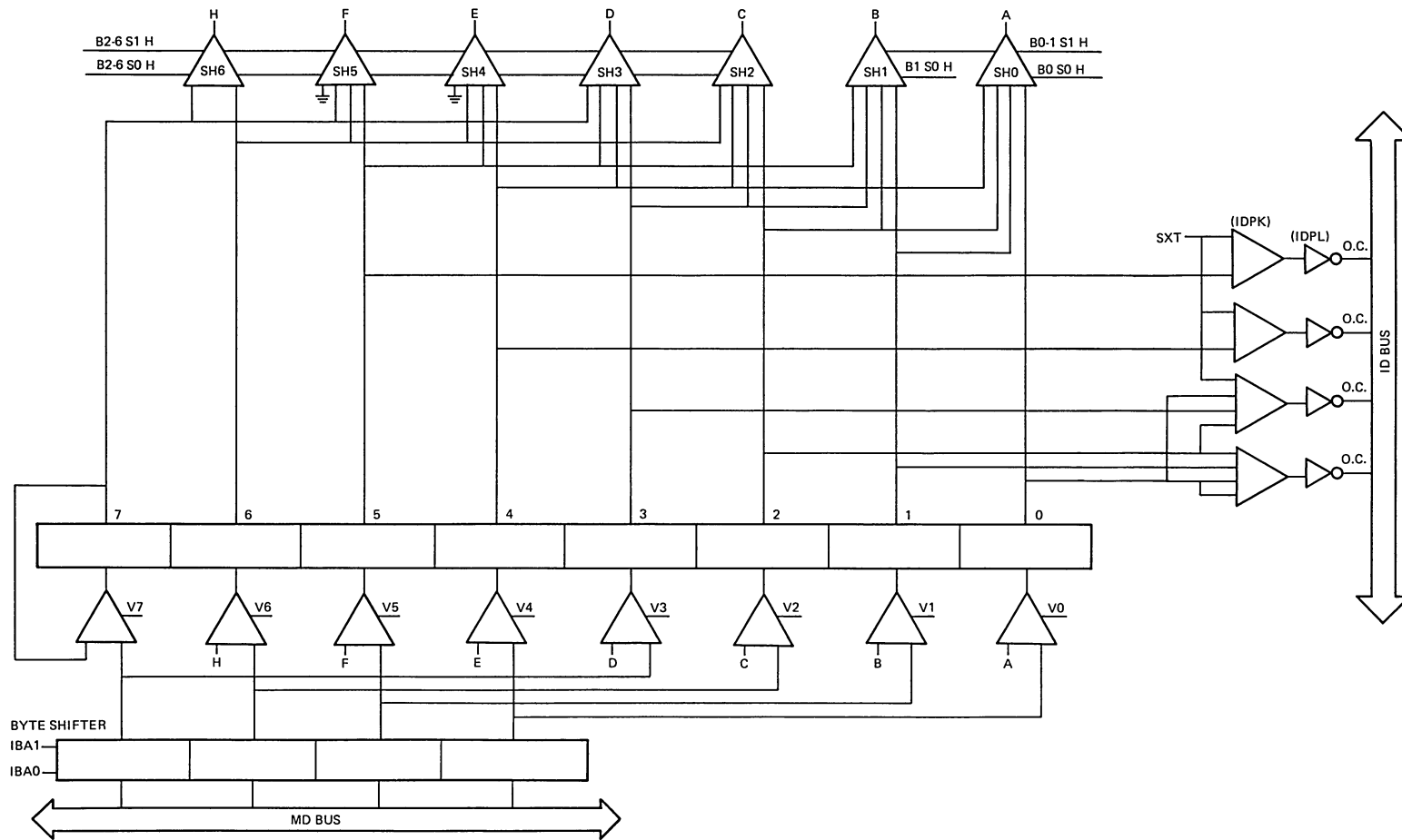
SBI/TB/Cache Block Diagram



BIBA 1	BIBA 0	RMD B0-0	RMD B1-0	RMD B2-0	RMD B3-0
S1	S0	Y3	Y2	Y1	Y0
LO	LO	MD0	MD01	MD02	MD03
LO	HI	MD1	MD2	MD3	MD0
HI	LO	MD2	MD3	MD0	MD1
HI	HI	MD3	MD0	MD1	MD2

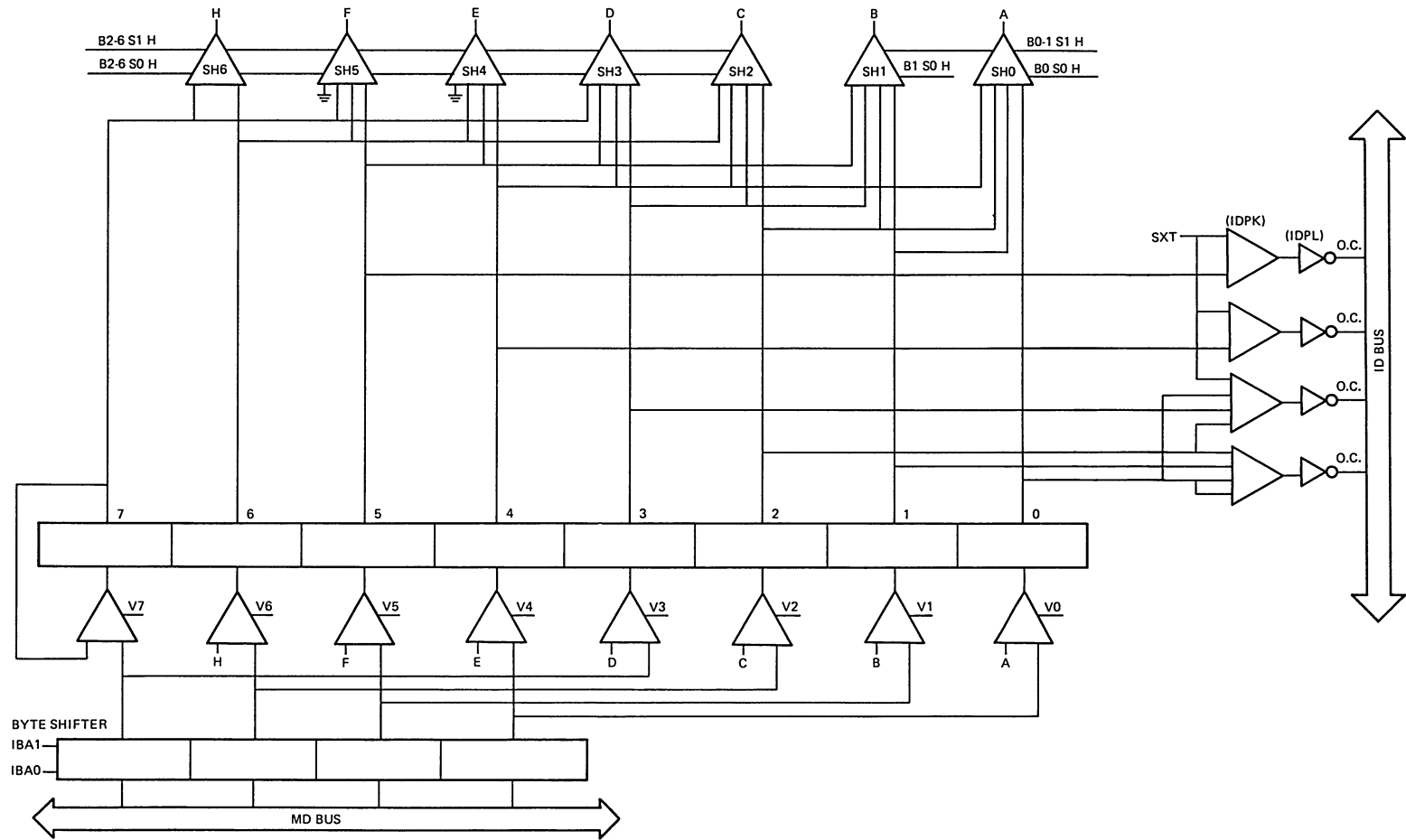
TK-4334

IDPN MD Byte Rotator



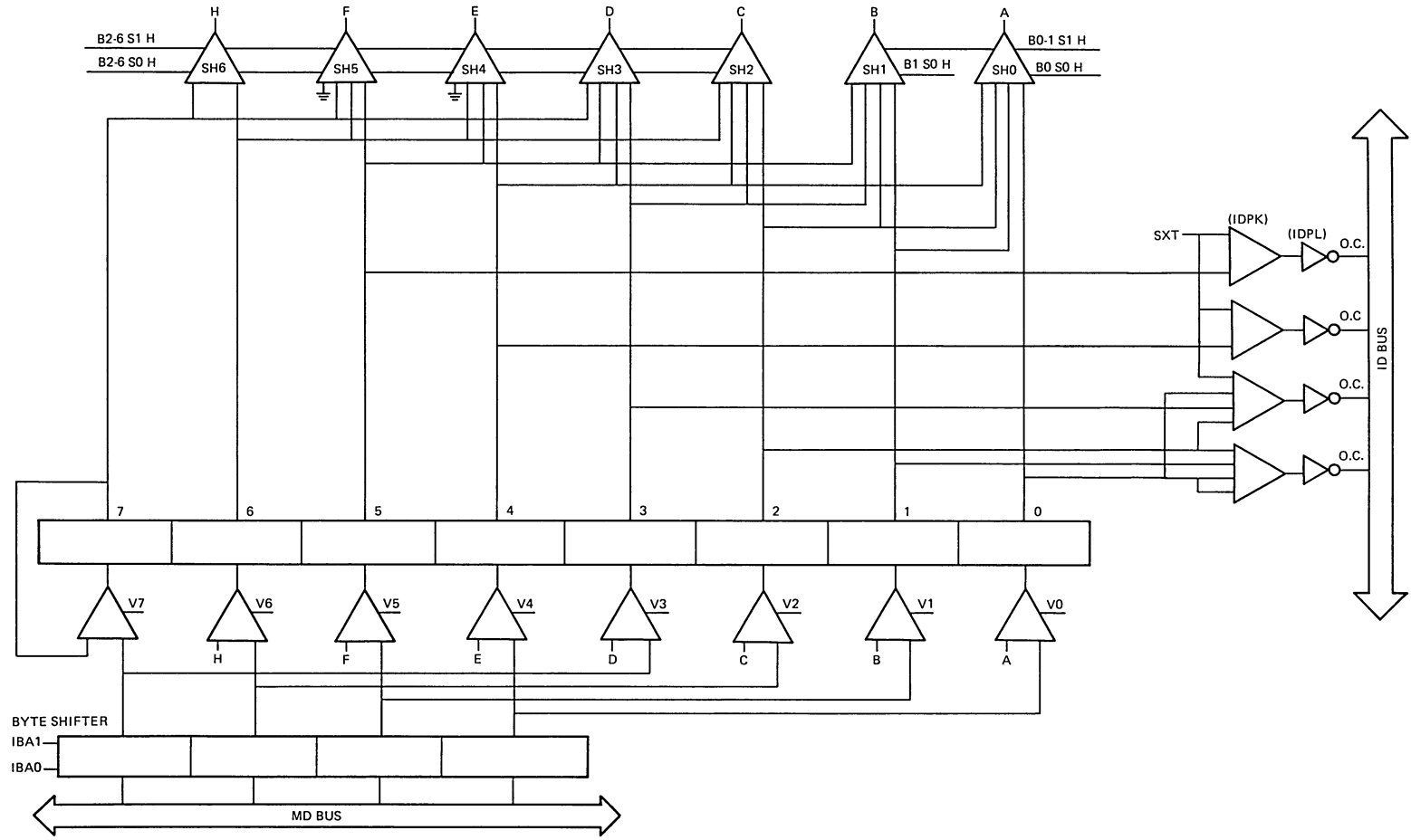
TX-4332

Instruction Data Path A



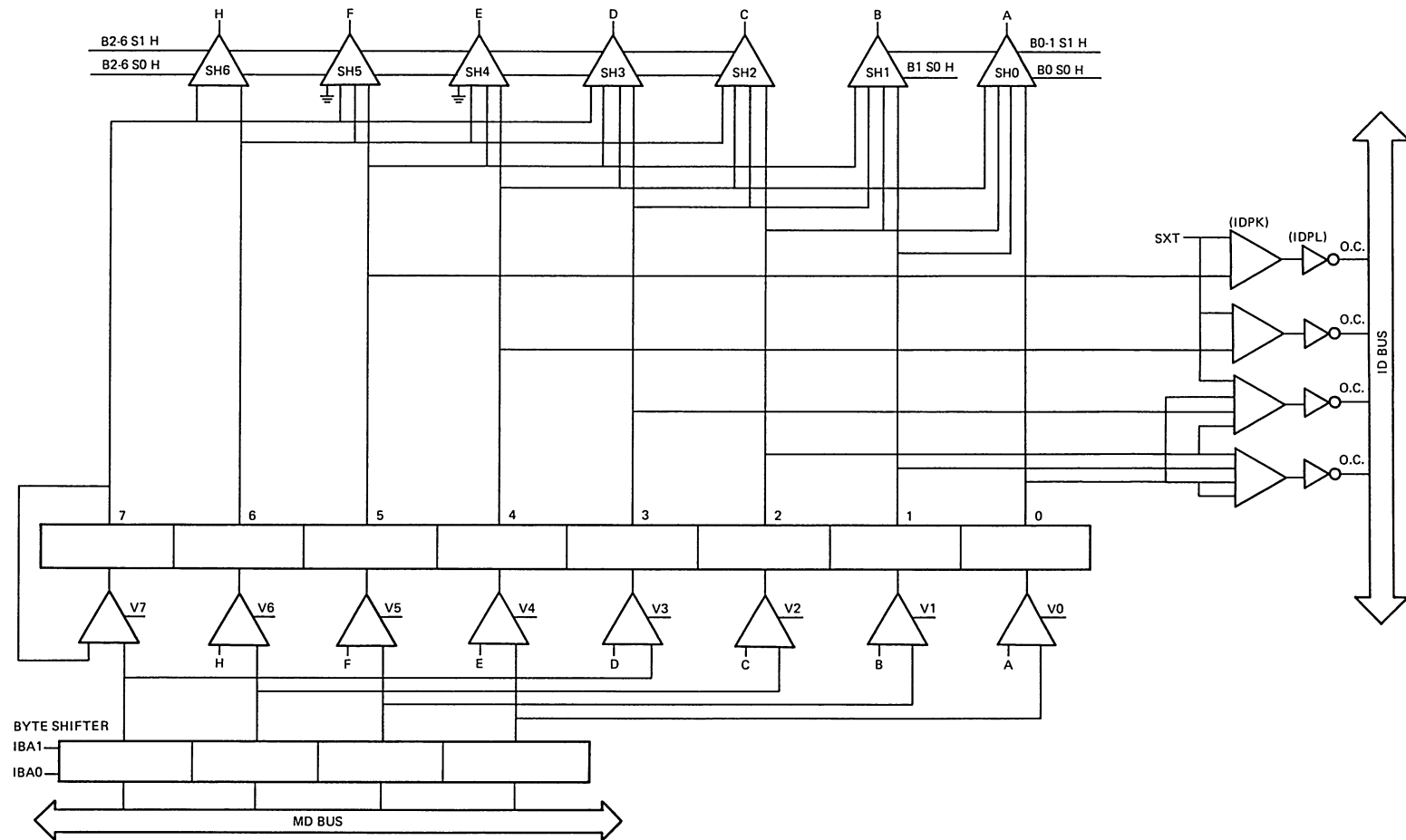
Instruction Data Path A

TK-4332



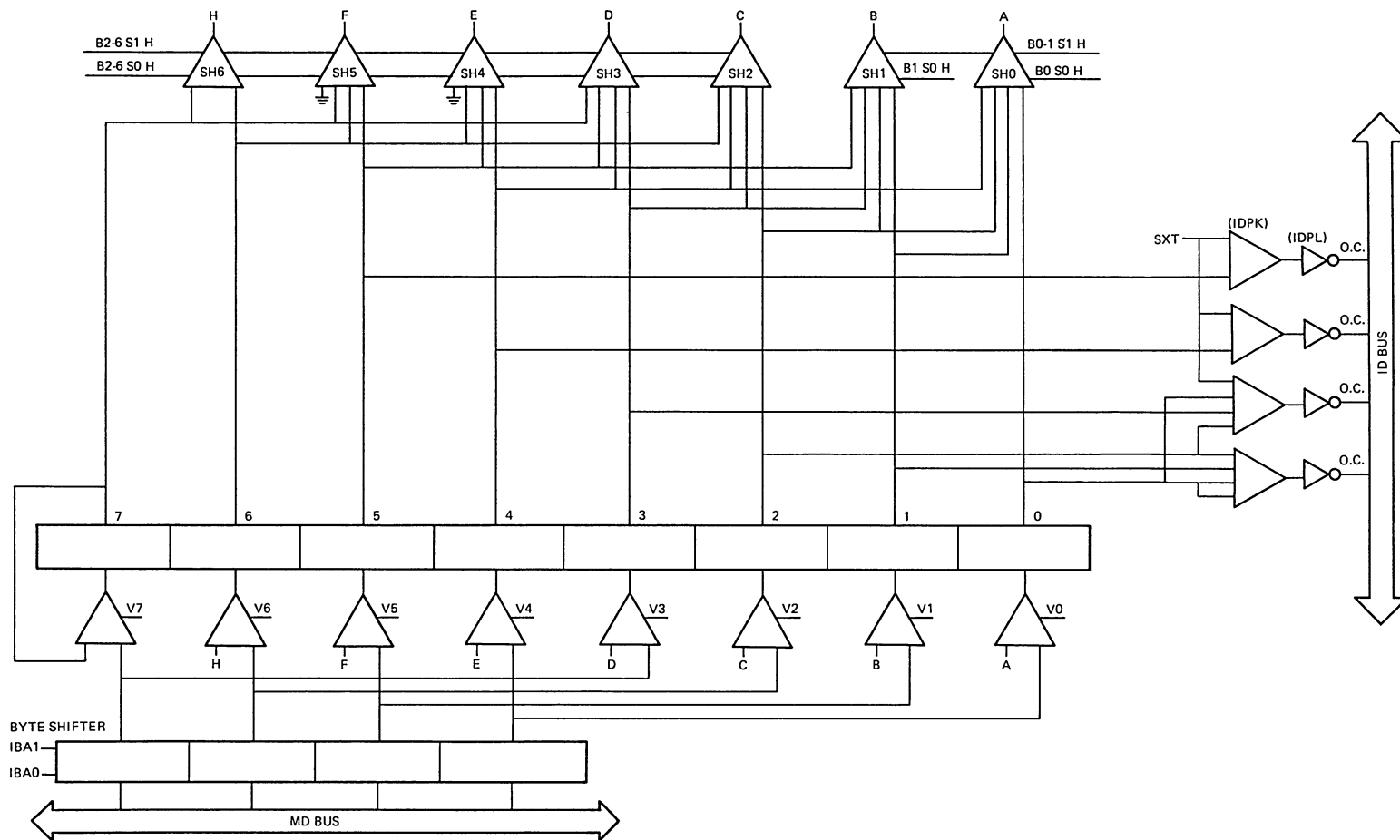
TK-4332

Instruction Data Path A



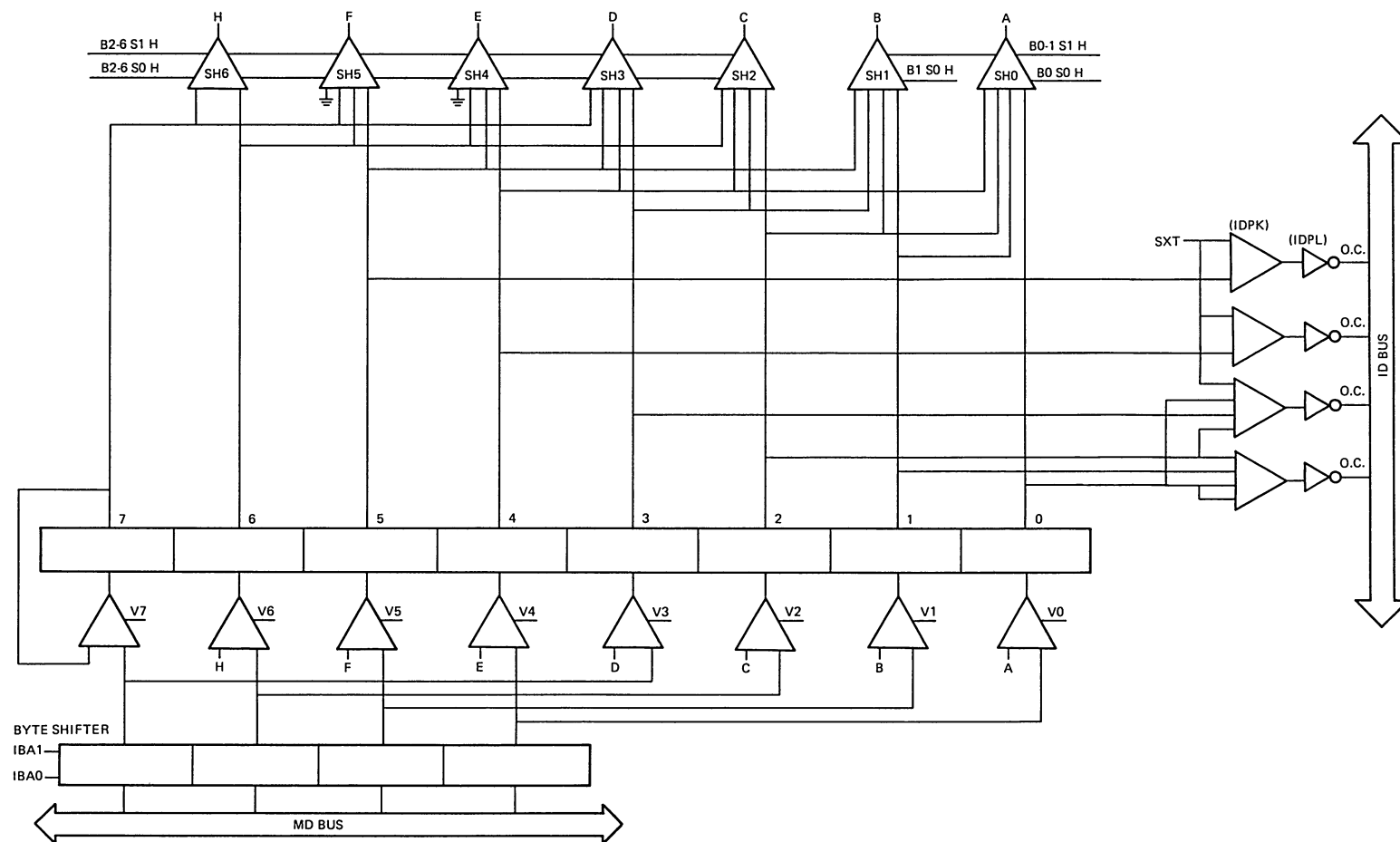
TK-4332

Instruction Data Path A



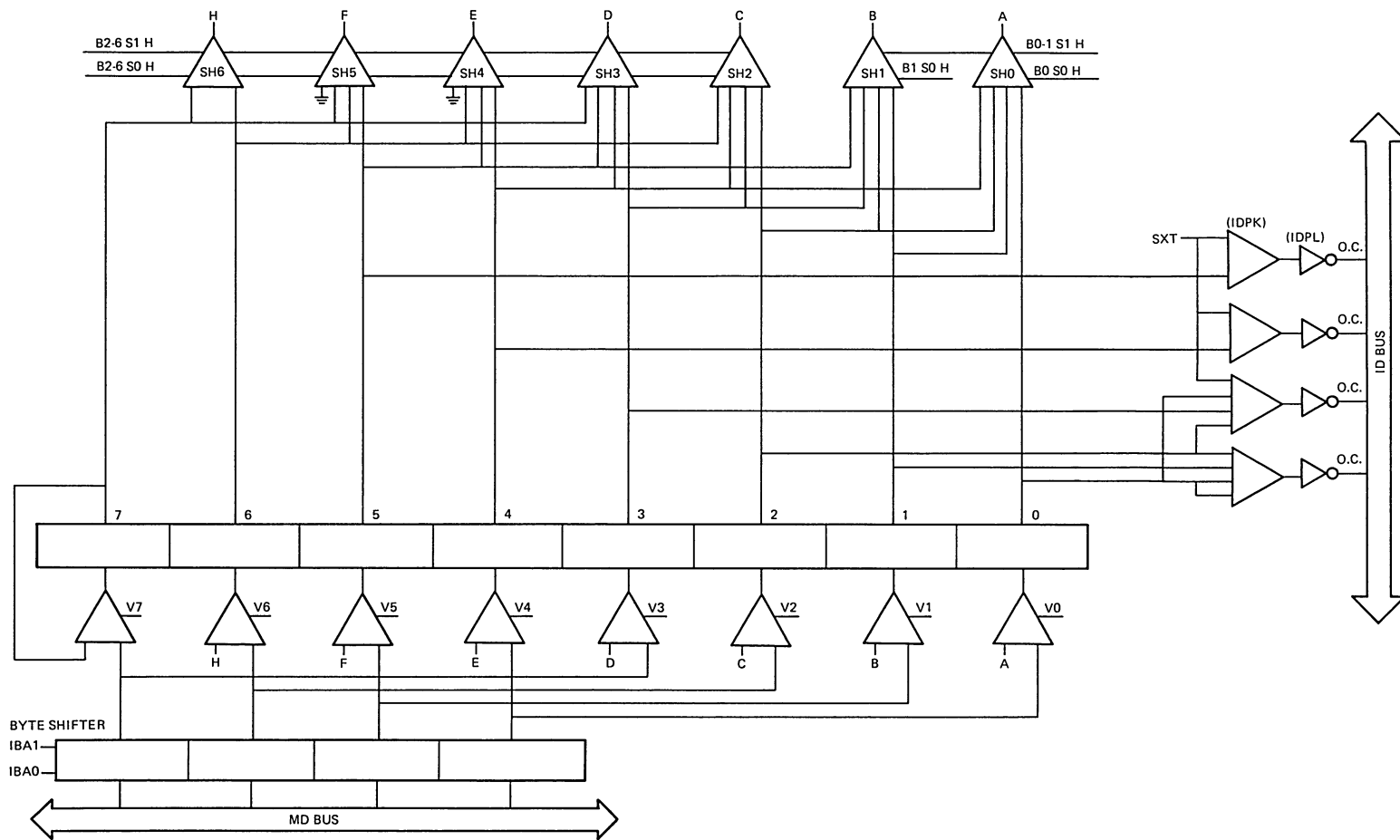
TK-4332

Instruction Data Path A



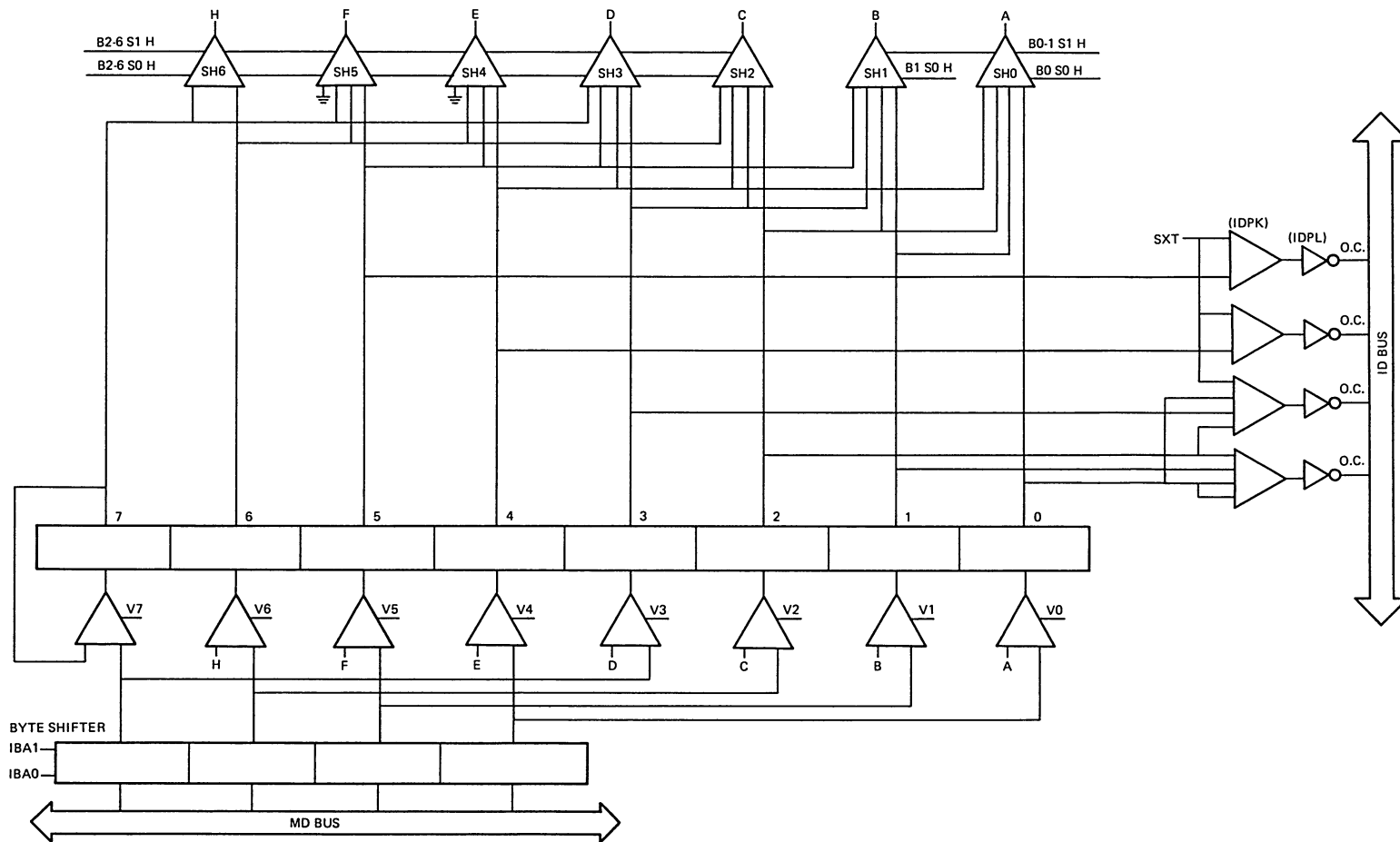
TK-4332

Instruction Data Path A



TK-4332

Instruction Data Path A



TK-4332

Instruction Data Path A

LAB EXERCISES

1. Acquire the correct floppy diskette from the instructor.
2. Using the indirect console command file procedure, load the lab exercise program. The instructor will supply the name of file.

EXERCISE A

The first run will prove the integrity of the program. The program will start using the Load Context instruction.

The process control block (PCB) is specified by the privileged register process control block base. The general registers are loaded from the PCB. The memory management registers describing the process address space are also loaded, and the process entries in the translation buffer are cleared. Execution is switched to kernel stack. The PC and PSL are moved from the PCB to the stack and are suitable for use by a subsequent REI instruction.

The scheduler will be invoked with a `CONSOLE CONTINUE` at a designated system space with the memory management bit enabled. In exercise J of this lab, you will be required to invoke the scheduler without memory management enabled and use the `CONSOLE START` command using a physical address. Turning on memory management will be done within the scheduler routine.

Perform the following steps to start the scheduler:

```
Step 1. >>>D PC 80000050      /system address of mini scheduler
Step 2. >>>C                  /CONSOLE CONTINUE
```

Upon a successful halt, examine R5 for the correct results.

The halted address is _____
 Does the halt address agree with lecture?
 If no, see instructor.
 If yes, continue.

R5 = _____. Does it agree with lecture?

If yes, then continue; if no, see instructor.

This next exercise duplicates the procedure we discussed in the classroom: running the program from `CONSOLE CONTINUE` command using virtual address 400.

EXERCISE B

Now that the program has proven itself, perform the following. Remember, the reason this program can now begin with a `CONSOLE CONTINUE` is because all the hardware registers were loaded by executing a Load Context instruction.

```
Step 1. >>>D/I 39 0      /invalidates all the translation buffer
Step 2. >>>D PC 400
Step 6. >>>C              /console continues
```

Upon successful halt (same halt address as in exercise A), examine R5.

Does R5 contain the same answer as found in exercise A. YES or NO?

R5 = _____. Is it the correct value? YES/NO. Explain:

EXERCISE C

In the lecture, the instructor discussed the difference between `CONSOLE START` and `CONSOLE CONTINUE`. To prove this discussion, you will set up a series of short experiments from the console.

Experiments 1-3 will deal with `CONSOLE START`.
Experiments 4-6 will deal with `CONSOLE CONTINUE`.

Deposit into memory a simple routine:

```
>>>D/ID 12 0      /turns memory management off
>>>D 400 00528190 /MOVB (R1)+, R2
                  /HALT
```

The contents of the registers are not meaningful at this time.

Experiment 1. Pick a ROM state (not 100) that is in the PCS initialize routine (refer to PCS listing). Set up the proper registers to stop on micromatch.

Selected ROM state _____
Perform >>>S 400

The 11/780 should have stopped with the micromatch, proving that `CONSOLE START` enters the PCS initialize sequence. If the stop on micromatch did not occur, check with the instructor; otherwise, continue with the lab.

Clear up this condition and start experiment 2. (Clear up this condition means to bring the 11/780 back to ROM state 0FF with the clock running.)

Experiment 2. Pick a ROM state (not 123) that is in the PCS deposit general register routine (refer to PCS listing). Set up the proper registers to stop on micromatch.

Selected ROM state _____
Perform >>>S 400

The 11/780 should stop on micromatch, once again proving that CONSOLE START enters the PCS sequence to deposit general registers.

If it does not, see the instructor for aid; otherwise, continue.

Clear up this condition and begin experiment 3.

Experiment 3. Using ROM state 127, set up the proper registers to stop on micromatch.

Perform >>>S 400

The 11/780 should stop on micromatch, proving that CONSOLE START enters the PCS sequence for CONSOLE CONTINUE.

If it does not, see the instructor for aid; otherwise, continue.

Clear up this condition and start the CONSOLE CONTINUE experiments.

Experiment 4. Select the same ROM state that you picked in experiment 1. Set up the proper registers to stop on micromatch.

Perform >>>D PC 400
>>>C

The 11/780 should not have stopped but should have continued on to the halt address 403. This proves that CONSOLE CONTINUE does not enter the PCS sequence for initialize.

If it does not, check with instructor and clear up this condition; otherwise, continue.

Experiment 5. Before selecting the same ROM state as you did in experiment 2, perform
>>>D PC 400.
Then set up the proper registers to stop on micromatch.
Perform
>>>C

The 11/780 should not have stopped but should have continued on to halt address 404. This proves that CONSOLE CONTINUE does not enter the PCS sequence to deposit general registers.

If it does not, check with instructor for aid and clear up this condition; otherwise, continue.

Experiment 6. Select ROM state 127 and set up the proper registers to stop on micromatch.
 Perform >>>D PC 400
 >>>C

The 11/780 should have stopped on micromatch. Again, check with instructor if it did not.

Experiments 1 and 6 should prove that both CONSOLE START and CONSOLE CONTINUE enter ROM state 127. CONSOLE CONTINUE does not perform an initialize sequence.

Clear up this condition and continue.

In the next series of steps, you will analyze the PSL when the initialize sequence is implemented by microcode.

Perform the following:

Step 1. >>>D/ID F 0 /clears out PSL
 Step 2. >>>E/ID F /examine PSL
 Step 3. Record contents of PSL _____
 Step 4. >>>I /implements Init routine
 Step 5. >>>E/ID F /examine PSL
 Step 6. Record contents of PSL _____

What was the difference, if any, noted between step 3 and step 6?

Two types of faults are associated with memory mapping and protection. A translation-not-valid fault occurs when a read or write reference is attempted through an invalid PTE. An access control violation fault occurs when the protection field of the PTE indicates that the intended access to the page for the specified mode would be illegal.

Reload the program from diskette at this time and suppress the printout by typing Control O. Invoke the scheduler again, to load the PCB for this process.

```
>>>D PC 80000050
```

```
>>>C
```

Proceed upon a successful halt.

EXERCISE D

In this exercise, you will examine a page table entry that is invalid. You will see how the program behaves under normal operating conditions, which are: if there is a PTE invalid, the 11/780 will vector to 24 if the SCBB is 0 (which it should be), and 24 should point to the address to handle this type of fault. The fault handler will validate the PTE and also move a code to R8 to verify a successful completion and return.

```
Step 1. >>>D/I 39 0           /invalidates TB
Step 2. >>>D A08 20000001      /invalid PTE
Step 3. >>>E R8                /R8 = _____
Step 4. >>>D PC 400
Step 6. >>>E SP                /SP = _____
Step 7. >>>C
```

The program should run to the same halt address recorded in A. Perform the following examines to prove the fault occurred and was corrected.

```
Examine R8           contents = _____
```

```
Examine SP           contents = _____
```

The stack pointer should have returned to its original value because of the stack clean-up routine.

Was the PTE corrected in memory? _____

To answer the next series of questions pertaining to the parameters pushed on the stack during an invalid PTE, perform the following:

```
>>>Q 0/N:1FFF          /clears memory
```

Then, reload the program from diskette, use Control 0 to suppress the printout, and load the PCB:

```
>>>D PC 80000050
```

```
>>>C
```

When the program halts, perform the following:

```
>>>D/I 39 0          /invalidates the TB
>>>D A08 20000001    /invalidates the PTE for operands
>>>D/B 808 00        /substitute a halt to prevent stack clean up
>>>D PC 400
>>>C
```

The program now halts in system space. This halt, used as a breakpoint, will allow you to analyze the stack.

How many pushes were on the stack? _____

Was it the correct amount? _____

Were the correct parameters pushed on the stack? _____

What reference material did you use to verify what parameters are pushed on the stack? _____

Was the correct stack pointer used? _____

Even with an invalid PTE, the run time was not noticeably different, which would normally be the case. There are different approaches to prove that this fault occurs. One is to deposit a trap catcher in a physical location 24. Another is the one you just performed by inserting a halt breakpoint. Another method is to pick a microstate that you know is part of the PTE invalid routine within PCS and set up all the proper registers required so the 11/780 will stop on a micromatch. Do not forget to set up the 11/780 to guarantee the condition you are trying to duplicate.

Selected ROM state _____

If you have difficulty performing this stop on micromatch, check with the instructor. You must clear this condition before proceeding.

We will now consider access control violation faults. Reload the program from diskette and suppress the printout by typing Control O. Invoke the scheduler to load the PCB for this process.

```
>>>D PC 80000050
>>>C
```

Proceed after a successful halt.

Since the basic fault routine is limited in scope, the PTE that will be corrected is physically located at A08.

EXERCISE E

Observe the program's behavior under normal operation with an access violation and perform the following:

```
>>>D/I 39 0
>>>D A08 80000001      /no access protection code
>>>D PC 400
>>>E R9                /R9 = _____
>>>C
```

The 11/780 should have halted with the same address as noted in experiment 1. Note also that the exception handler for an access violation contains a Save Context instruction.

The process control block is specified by the privileged register process control block base. The general registers are saved in the PCB. The PC and PSL on the top of the current stack are popped and stored in the PCB. If an SVPCTX instruction is executed when IS is clear, then IS is set, the interrupt stack pointer is activated, and IPL is maximized with a 1 because of the switch to the interrupt stack. The map, ASTLVL, and PME contents of the PCB are not saved.

The old PCB should have been saved starting at what physical location? Physical address _____

How many locations were loaded by the SVPCTX instruction? _____

Within that same routine, R9 should have been loaded with a number to prove a successful transfer to the exception handler.

R9 contains _____

The stack pointer should also be examined at this time.

SP contains _____

Once again, the stack pointer should have been restored because of the stack clean-up routine.

Compare the PCB that was saved to the PCB starting at physical location 1200. Make a note of any differences.

To answer the next series of questions pertaining to the parameters pushed on the stack during an access violation, perform the following:

```
>>>Q 0/N:1FFF
```

Then reload the program from diskette, use Control O to suppress the printout and load the PCB:

```
>>>D PC 80000050
>>>C
```

After a successful halt, alter the exception routine by placing a breakpoint halt in an advantageous location. Then perform

```
>>>D/I 39 0
>>>D A08 80000001
>>>D PC 400
>>>C
```

The program halts in system space. This halt used as a breakpoint will allow you to analyze the stack.

How many pushes were on the stack? _____

Was it the correct amount? _____

Were the correct parameters pushed on the stack? _____

Were the parameters different for an invalid PTE? _____

At the console, you probably did not notice any time lapse during an access violation. Once again, pick a ROM state in the PCS sequence of an access violation and stop on micromatch.

Selected ROM state _____

If you have difficulty performing this stop on micromatch, check with the instructor. Clear the condition before proceeding.

EXERCISE F

Locate a ROM state within the PCS sequence that performs LDPCTX instruction. (The Load Context instruction is used to load the hardware registers necessary to run the program, as in Exercise A.)

Set up the the 11/780 to stop on micromatch.

Selected ROM state _____

Run the program again, this time with a `CONSOLE CONTINUE`. After the micromatch occurs, step through a few of the ROM states to verify your understanding of the PCS sequence. Once you feel comfortable, clear up the condition and proceed to exercise G. Feeling comfortable means that not only can you select any ROM state in the PCS sequence, but you can also predict 100 percent of the time the next ROM state within the sequence by using the comments within the PCS listing and the knowledge acquired from the classroom lecture.

EXERCISE G

Run the program, again with a `CONSOLE CONTINUE`, but pick a ROM state that is in the REI PCS sequence.

Selected ROM state _____

After the micromatch occurs, step through a few of the ROM states to verify your understanding of the PCS sequence. Once you feel comfortable, clear up the conditions and proceed to the next exercise.

EXERCISE H

Run the program with a `CONSOLE CONTINUE` but pick a ROM state that is in the ADDB3 PCS sequence.

Selected ROM state _____

After the micromatch occurs, step through a few of the ROM states to verify your understanding of the PCS sequence. Once you feel comfortable, clear up the conditions and proceed.

In the next exercise, you will analyze the access violation using the current mode of the PSL and protection code of the PTE.

If the PSL contains 03000000, then 11/780 is in user mode.
 If the PSL contains 02000000, then 11/780 is in supervisor mode.
 If the PSL contains 01000000, then 11/780 is in executive mode.
 If the PSL contains 00000000, then 11/780 is in kernel mode.

EXERCISE I

This exercise may be redundant, but it will give you more insight to the protection provided by the software and how the hardware implements these checks.

At the beginning of each experiment in this group, you will perform the following steps.

Step 1. >>>Q 0/N:1FFF /clears memory

Step 2. Load the program from diskette using the indirect command file procedure. Load the hardware registers by the following:

```
>>>D PC 80000050
>>>C
```

Step 3. Because of the limited exception handler routines, load trap catchers in physical locations 20 and 24.

```
>>>D 20 3
>>>D 24 3
```

Step 4. Insert the new PTE at physical location A08.

```
>>>D A08 90000001
```

Decode the protection code in the new PTE and state what the protection is now. What was the protection for the old PTE?

Experiment 7. Perform Steps 1-4, then:

```
>>>D/I 39 0                   /invalidates TB
>>>D PC 400
>>>D PSL 03000000           /user mode in PSL
>>>C
```

After `CONSOLE CONTINUE` was performed, the console should have responded with

```
?ILL I/E VEC
HALTED AT 00000020
```

If it did not, then see the instructor; otherwise, continue with the analysis.

How many pushes were on the stack? _____
 What stack was used? _____
 Was it the correct stack for this type of exception? _____
 Were the correct parameters pushed on the stack? _____
 What reference material can you use to verify these parameters?

Experiment 8. Perform Steps 1-4, then:

```
>>>D/I 39 0          /invalidates TB
>>>D PC 400
>>>D PSL 02000000    /supervisor mode
>>>C
```

After CONSOLE CONTINUE was performed, the console should have responded with

```
?ILL I/E VEC
HALTED AT 00000020
```

If it did not, then see the instructor; otherwise, continue with the analysis.

How many pushes were on the stack? _____
 What stack was used? _____
 Was it the correct stack for this type of exception? _____
 Were the correct parameters pushed on the stack? _____
 What reference material can you use to verify these parameters?

Did the parameter for this access violation in supervisor mode differ from the access violation in user mode?

Experiment 9. Perform Steps 1-4, then:

```
>>>D/I 39 0          /invalidates TB
>>>D PC 400
>>>D PSL 01000000    /executive mode in PSL
>>>C
```

After CONSOLE CONTINUE was performed, the console should have responded with

```
?ILL I/E VEC
HALTED AT 00000020
```

If it did not, then see the instructor; otherwise, continue with the analysis.

How many pushes were on the stack? _____
 What stack was used? _____
 Was it the correct stack for this type of exception? _____
 Were the correct parameters pushed on the stack? _____
 What reference material can you use to verify these parameters?

Did this access violation differ from the previous two?

Now select the ROM state within the PCS sequence that performs this check between the PSL and PTE for an access violation.

Selected ROM state _____

EXERCISE J

Return the program to its original state by reloading the program using the indirect command file from the console. By now you should have enough background to incorporate a length violation and modify the exception handler routine to correct the situation. Examine the stack parameters for a length violation. Does it differ from an access violation? Keep this program simple.

EXERCISE K

Modify this program so physical address 850 can be used in conjunction with a CONSOLE START. In other words,

```
>>>S 850
```

will invoke the scheduler while the memory management is not enabled. Memory management should be enabled within the scheduler routine. Keep this program simple.

EXERCISE L

Return the program to its original state by reloading the program using the indirect command file from the console. Perform the following:

1. >>>D 124C 02800000 /PSL is in supervisor mode
2. >>>D PC 80000050
3. >>>D 20A 005C8FBB /PUSHR
4. >>>D/B 20E 00 /HALT
5. >>>C

The response to CONSOLE CONTINUE should be:

```
?ILL I/V VEC
HALTED AT 00000010
```

If this response did not occur, check with the instructor.

If the proper response did occur, perform the following:

>>>E/I 0/N:4

After the required printout, answer the following questions.

1. What is the content of the kernel stack point? _____
2. What is the content of the supervisor stack pointer?

3. Contents of what registers were pushed on the supervisor stack? _____, _____, _____, _____. Do these registers agree with the PUSHHR instruction mask? _____
4. What type of violation occurred?
5. Were the correct parameters pushed on the kernel stack?

Research questions

1. What type of exception is taken if a halt instruction is implemented when the PSL is in user mode? Can you prove your reply with the program under study?

IF YES - How?

IF NO - Why not?
2. Start the program in kernel mode as usual, except that the PSL located in the hardware PCB is in user mode. Set up the PSL correctly so the program will run until the Halt instruction is encountered and a trap to location 10 will be invoked for trying to perform a privileged instruction.
3. Does a Save Context instruction save the whole PCB?

BLANK

MODULE TEST

1. Given the instruction `MOVW (R1)+, R2`, the next entry microword address into PCS after IRD (062) is:
 - a. 008
 - b. 009
 - c. 094
 - d. 00A

2. Given the instruction `MOVL #^X2000, R4`, the correct microsequence in PCS to execute the entire instruction is:
 - a. 062
019
065
224
 - b. 062
01B
065
224
 - c. 062
084
065
224
 - d. 062
00D
065
224

3. Given the instruction `CLRW ^X2000(R4)`, the first three microword addresses of the PCS sequence after IRD (062) are:
 - a. 004F
027E
00D0
 - b. 004F
027E
030D
 - c. 004F
027E
020D
 - d. 004F
00D0
027E

MODULE TEST

4. Given the instruction `ADDW3 #^X20, @#^X200, R5`, the first three microword addresses in the PCS sequence after IRD (62) are:

- a. 018
065
224
- b. 019
064
27E
- c. 000
21B
0D4
- d. 000
0D4
21B

Match the entry points (column A) of the PCS with the correct event (column B) for that entry point into PCS.

Column A

Column B

- 5. ____ 105
- 6. ____ 060
- 7. ____ 062
- 8. ____ 127
- 9. ____ 038
- 10. ____ 100
- 11. ____ 102
- 12. ____ 104
- 13. ____ 10E
- 14. ____ 10D
- 15. ____ 10F
- 16. ____ 10C
- 17. ____ 101

- a. System initialization
- b. Unalign trap
- c. Internal interrupt
- d. Translation buffer miss
- e. Control store parity error
- f. External interrupt
- g. IRD state
- h. Odd address error
- i. Translation buffer parity error
- j. Console continue
- k. Page trap
- l. Instruction buffer miss
- m. Timeout
- n. Read data substitute
- o. Modify bit
- p. Reserved float operand
- q. Cache parity error
- r. Protection violation

MODULE TEST

Questions 18-20 will use the following PCS sequence:

0062
0062
0062
0062
0062
0062
0062
000D
0094
024E
024E
024E
024E
024E
024E
024E
024E
024E
0341
0062

18. Identify one microword address that is in B-fork logic.

- a. 094
- b. 24E
- c. 341

19. Cache stall is obvious. Describe in your own words how the microword address remains unchanged in the microsequence. You should use some reference to the functional blocks necessary and any logic circuits required.

20. Entry point 00D is used for which mode of operation?

- a. Deferred
- b. Index
- c. Displacement
- d. Short literal

BLANK

WCS DEBUGGER

INTRODUCTION

This lesson will introduce you to WCS and WCS debugger. Depending on the situation, WCS debugger can be used by the technician as a maintenance tool to help diagnose errors that are detected, but not isolated, by the the microdiagnostic and macrodiagnostics.

OBJECTIVES

1. Using WCS debugger, develop and enter into the machine a small (up to 5 instructions) microroutine to perform specified functions.
 - a. Arithmetic/logic functions of the ALU
 - b. Data manipulation in the data path
 - c. Switch between WCS and PCS microcode
 - d. Deposit/examine data from memory
2. Given a failing WCS microroutine, issue WCS commands to debug the routine.

6

SAMPLE TEST ITEM

This is a performance test to evaluate your understanding of the objectives. The instructor will evaluate the printout that you obtain during this test.

1. Using WCS debugger, enter into WCS a routine that simulates the performance of the following instructions.
 - a. CLRB R1 R1 initially contains FFFFFFFF

RESOURCES

1. WCS Monitor Help File

LECTURE OUTLINE

- I. Why WCS?
- II. Ways to Write into WCS
- III. WCS Debugger
- IV. Macroprogram
- V. Console Mode

XFC OP CODE PROGRAM

This program is designed to illustrate the usage of the XFC op code with a customer-created instruction running out of the VAX WCS. This particular instruction reads a number of memory locations specified in the instruction and adds the contents of each location to the previous location. When the addition is complete the results are stored in a register that is specified by the instruction. The format of the instruction in memory is,

SP3 SP2 SP1 FC

where specifier 1 is the scratch pad address of the GPR that contains the number of memory locations to be added. SP2 is the first memory location to be added. SP3 is the scratch pad address of the register that will hold the result of the addition. The specifiers can contain any register number from 0 to 15.

In this example,

R1 = no. of memory locations

R2 = virtual address of first location

R3 = result

```

0000 18      .Title XFC
0000 19
0000 20      .MACRO MEMADD A, R, C      ;this macro will format
0000 21      XFC                      ;XFC opcode and specifiers
0000 22      .FYTE 'A                  ;for execution
0000 23      .BYTE 'B
0000 24      .BYTE 'C
0000 25      .ENDM MEMADD
0000 26
0000 27      .MACRO LDWCS A, H          ;this macro is used to load
0000 28      MOVL 'A, R0                ;the micro instructions from
0000 29      MOVAL 'B, R1              ;the uword table into the
0000 30      JSB WRWCS                 ;VAX WCS.
0000 31      .ENDM LDWCS
0000 32
00000011 0000 33      SCB=^X11          ;processor register definitions
0000002C 0000 34      WCSADD=^X2C
0000002D 0000 35      WCSDAT=^X2D
0000 36
0000 37      .REPT 128                 ;load trap catchers into vectors
0000 38      .LONG 3                  ;page (address 0 to 1FC)
00000003 0000 39      .ENDR
0200 40
00000200 0200 41      .=^X200
00000250 0200 42      .BLKL 20
0250 43      STACK:                   ;interrupt stack area
50 03 C4 0250 44 WRWCS: MULL2 #3, R0   ;WCS load routine called
2C 81 DA 0253 45      MTPR (R1)+, #WCSADD ;in the LDWCS macro
2D 81 DA 0256 46 1$: MTPR (R1)+, #WCSDAT
FA 50 F5 0259 47      SORGTR R0, 1$
05 025C 48      RSB
025D 49
5E F0 AF DE 025D 50 START: MOVAL STACK, SP ;initialize the stack
11 00 DA 0261 51      MTPR #0, #SCB      ;set vector page address = 0
0264 52      LDWCS #5, UWORD            ;load the WCS (1009-100D)
0271 53      LDWCS #1, UWORD+64         ;load WCS address 100F
027E 54      LDWCS #1, UWORD+80         ;load WCS address 10E0
00000014 9F 02 D0 028B 55      MOVL #2, R#^X14 ;put 2 in the customer vector
51 0A D0 0292 56      MOVL #10, R1      ;put test values in registers
52 00002000 8F D0 0295 57      MOVL #^X2000, R2

```

```

55  02  D0  029C  58      MOVL #2, R5
    53  D4  029F  59      CLRRL R3
    00  02A1  60      HALT                      ;halt to examine or modify
54  51  D0  02A2  61      MOVL R1, R4          ;put a pattern in memory for
56  52  D0  02A5  62      MOVL R2, R6          ;the test.
86  55  D0  02A8  63  2s:  MOVL R5, (R6)+
    FA 54  F5  02AB  64      SOBGTR R4, 2s
    00  02AF  65      HALT                      ;halt to single bus cycle if
    00  02AF  66      MEMADD 1, 2, 3          ;desired and execute XFC
    00  02B3  67      HALT
    00001009 02B4  69  UWORD: .LONG ^X00001009      ;WCS address 1009
    00001009 02B8  70
    0182100A 02B8  71      .LONG ^X0182100A      ;FE < SC, SC < 0
    00008400 02BC  72      .LONG ^X00008400      ;allow IB read.
    0001203C 02C0  73      .LONG ^X0001203C
    0001203C 02C4  74
    0200100B 02C4  75      .LONG ^X0200100B      ;VA < SP2 register data
    00008440 02C8  76      .LONG ^X00008440      ;allow IB read
    D000003C 02CC  77      .LONG ^XD0000003C      ;clear IB byte 1
    0000003C 02D0  78
    0080D00C 02D0  79      .LONG ^X0080D00C      ;VA < VA+4
    00004803 02D4  80      .LONG ^X00004803      ;SC < SC+1
    00000000 02D8  81      .LONG ^X00000000      ;D < cache long
    00000000 02DC  82
    0010B00D 02DC  83      .LONG ^X0010B00D      ;0 < 0 + D
    00400000 02E0  84      .LONG ^X00400000      ;SC = FE, Set EALU CC
    001D2014 02E4  85      .LONG ^X001D2014
    0000100B 02E8  86
    000000C0 02E8  87      .LONG ^X0000100B      ;SP3 register < 0
    000000C0 02EC  88      .LONG ^X000000C0      ;EALU=Z?, yes go to 100F
    0001323C 02F0  89      .LONG ^X0001323C      ;no, go to 100B
    0000100F 02F4  90
    0000100F 02F8  91      .LONG ^X0000100F      ;WCS address 100F
    00000062 02F8  92
    00000006 02F8  93      .LONG ^X00000062      ;PC < PC+4
    00000006 02FC  94      .LONG ^X00000006      ;IB clear byte 0 and 1
    40000000 0300  95      .LONG ^X40000000      ;jump to IRD (62)
    000010E0 0304  96
    000010E0 0308  97      .LONG ^X000010E0      ;WCS address 10E0
    000010E0 0308  98
    00821009 0308  99      .LONG ^X00821009      ;SC < SP1 register data
    00788440 030C 100      .LONG ^X00788440      ;0 < 0, IB clear byte 1
    D00C0038 0310 101      .LONG ^XD00C00038      ;allow IB read.
    0314 102
    0314 103      .END START

```

LAB EXERCISES

This lab is not intended to make you a microprogrammer but to fortify your understanding of the microword field definitions and their functionality within the VAX CPU. These exercises are designed to make use of most of the reference materials and instructor notes that are available.

The following exercises should be done with the aid of WCS debugger. Any time you need help, ask the instructor.

Write a sequence of microword routines that will perform the following functions:

1. RA(4) + RC(3) store results in RA(4)
2. RA(5) + RC(3) store results in RA(2)
3. RA(4) exclusive ORed RC(3) store results in D register.
4. RA(2) - RC(4) store results in Q register.
5. RA(5) - 2 the results are stored into RA(5). The ALU function must be A-B(Rlog). Examine the Rlog under DEBUGGER for proper answer.

In the next group of exercises, simulate, in your own way, the following machine instructions. Use the debugger commands to deposit to registers as required.

- | | | | |
|------------------|-------------|-------------|------|
| 1. MOVW R1, R2 | R1/FFFF5000 | R2/AAAABBBB | |
| 2. MOVW R3, (R4) | R3/12345678 | R4/00004000 | |
| 3. MOVB (R5), R6 | R5/00000500 | 500/AAAAACB | R6/0 |

LAB EXERCISES

The last series of exercises will enable you to switch between PCS and WCS.

Deposit into memory using the debugger commands the following machine-language instruction.

52	51	D0	1000/	MOVL	R1, R2
		00	1003/	HALT	

R1/12348765

R2/BBBBBBBB

R3/00001000

With a series of microwords in WCS, set up all necessary registers such that when you leave WCS to an entry point in PCS, the short program will be executed.

After you have mastered these routines, study the documentation provided for writing WCS from macrocode, and under an operating system, attempt to write a similar program.

MODULE TEST

This is a performance test to evaluate your understanding of the module objectives.

The instructor will evaluate the printout you obtain during this test.

1. Using WCS debugger, you will enter into WCS a routine that will simulate in your own manner the performance of the following instructions.

- a. CLRB R1 R1 initially contains FFFFFFFF

- b. ADD R1, R4, R5 R1/5 R4/7 R5/0

Your printouts must contain the following data:

- a. All registers before and after
 - b. A printout of all the microwords necessary to complete your routine
2. Using WCS debugger, you will enter into WCS a routine that will simulate in your own manner the performance of the following function.

The function will increment five times and halt.

R1 initially contains zero (0).

Hint: Pick a WCS location best suited for the BEN logic,

MODULE TEST

3. Given the following WCS routine starting at location 1000:

WCS>E 1000

```
00001000  ACF=0 ACM=0 ADS=0 ALU=0 AMX=0 BEN=0 BMX=0 CCK=0
          CID=0 DK=0 DT=0 EAL=0 EBM=0 FE FS=0 IBC=C IEK=0
          UJM=0000 DMX=0 MCT=0 MSC=0 PCK=0 QK=0 RMX=0 SCO
          SGN=0 SHF=0 SI=0 SMX=0 SPO=00 USU=3 VAK=0
```

WCS>E 1001

```
00001001  ACF=0 ACM=0 ADS=0 ALU=0 AMX=0 BEN=0 SMX=0 CCK=0
          CID=9 DK=0 DT=3 EAL=0 EBM=0 FE FS=B IBC=0 IEK=0
          UJM=FF KMX=0 MCT=0 MDS=0 PCK=0 QK=0 RMX=0 SCO
          SNG=0 SHF=0 SI=0 SMX=0 SPO=00 USU=3 VAK=0
```

simulate the following instruction:

```
MOVW R1, R2
R1/FFFFACBD
R2/6789FFFF
```

This routine will not function as presented. Using WCS debugger, correct the routine so that it will perform correctly.

MICRODIAGNOSTICS

INTRODUCTION

This lesson will show you how the microdiagnostics perform error detection and report the failure. Also, you will become proficient at locating these diagnostic tests and error loops on the microfiche.

OBJECTIVES

1. Identify the function of each portion of the VAX-11/780 microdiagnostic system:
 - a. Hardcore monitor
 - b. Microtest monitor
 - c. Fail chain
 - d. Go chain
 - e. Hardcore test stream
 - f. Microtest stream
2. Given a VAX-11/780 microdiagnostic error message, interpret the results and take appropriate action.

7

SAMPLE TEST ITEM

1. CMPCA is a pseudoinstruction which is found in
 - a. Microtest stream
 - b. Hardcore test stream
 - c. both (a) and (b)
 - d. neither (a) nor (b)

RESOURCES

1. VAX-11/780 Diagnostic System Technical Description
2. Microfiche Library

LECTURE OUTLINE

- I. Microdiagnostic Program Overview
 - A. Two (2) major test divisions
- II. Hardcore Test Description
 - A. Hardcore test structure
 - B. Pseudoinstruction description
- III. Microtest Description
 - A. Microtest structure
 - B. Microtest description
- IV. Program Listings
 - A. Hardcore
 - B. Microtest
- V. Error Message Analysis

MODULE TEST

Match the definition in column A that best describes the term in column B.

Column A	Column B
___ 1. Both test division monitors are serviced by it.	A. Hardcore monitor
___ 2. Composed of a test stream of pseudo instructions.	B. Microtest monitor
___ 3. Consists of microtests loaded into WCS and executed at full speed.	C. Fail chain
___ 4. Reenters the failing microtest and begins fault isolation.	D. Go chain
5. CPMA is a pseudoinstruction which is found in	E. Hardcore test stream
a. Microtest stream	F. Microtest stream
b. Hardcore test stream	
c. both (a) and (b)	
d. neither (a) nor (b)	

This last question will be performed in a laboratory environment. The instructor will insert a fault that will generate an error message from the microdiagnostics.

Upon analyzing the results, you will identify the failing module and file possible logic areas within the module responsible for the error.

Attach the error printout and your analysis to this test sheet.

List the ROM states within the failing test that you are using to analyze the error. Also, list the five possible logic areas responsible for the error.

BLANK

FLOATING-POINT ACCELERATOR

INTRODUCTION

This lesson will present the operating characteristics and functional logic of the floating-point accelerator (FPA).

The FPA is an independent processor that processes in parallel with the base CPU to execute the standard floating-point instruction set.

OBJECTIVES

1. List the functions of the following signals:
 - a. FP Sync
 - b. CP Sync
 - c. Accelerator Override
2. Given a floating-point instruction, list the required microsequence needed to perform the instruction.
3. Given a number in floating-point format, identify the following:
 - a. Sign of the exponent
 - b. Sign of the fraction
 - c. Value of the exponent in base 10
 - d. Value of the fraction in base 10
4. Given a number in base 10, convert to the floating-point format as it would appear in:
 - a. Physical memory
 - b. CPU register
 - c. Data path of the FPA

SAMPLE TEST ITEM

1. The floating-point format 00004080 represents
 - a. -1
 - b. +1
 - c. -.1
 - d. +.1

RESOURCES

1. VAX-11/780 Floating-Point Accelerator Technical Description
2. FPA Print Set

LECTURE OUTLINE

- I. Introduction to FPA
- II. Physical Layout
- III. Introduction to Number Systems
- IV. Introduce FPA Block Diagram

BLANK

FLOATING-POINT ACCELERATOR

<u>Slot No.</u>	29	28	27	26	25	24
		M	M	M	M	M
		8	8	8	8	8
C		2	2	2	2	2
1		8	8	8	8	8
B		9	8	7	6	5
		FCT	FAD	FML	FMH	FNM

These are
in order.

Each board in the FPA usually executes a specific operation.

M8289 FCT

Microcode = 512 locations
48-bit microword

Contains FPA control (its
microsequence)
Sign processor
Exponent processor

M8288 FAD

Contains the fractional adder

M8286/87 FMH, FML

Contains the multiply logic low
and high bits. Note MULL
(integer multiply) is also
executed by this board.

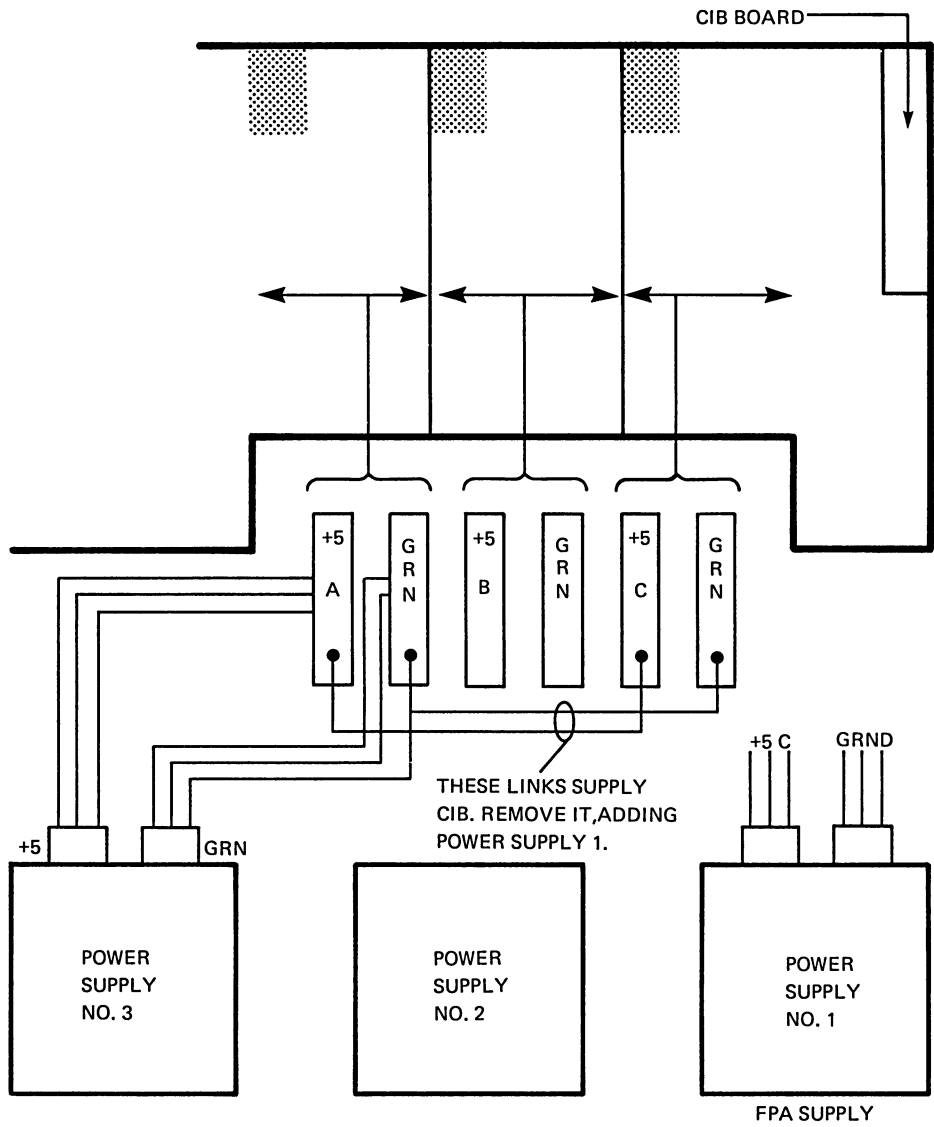
M8285 FNM

Contains the fraction
normalizer and divide logic

FPA Installation

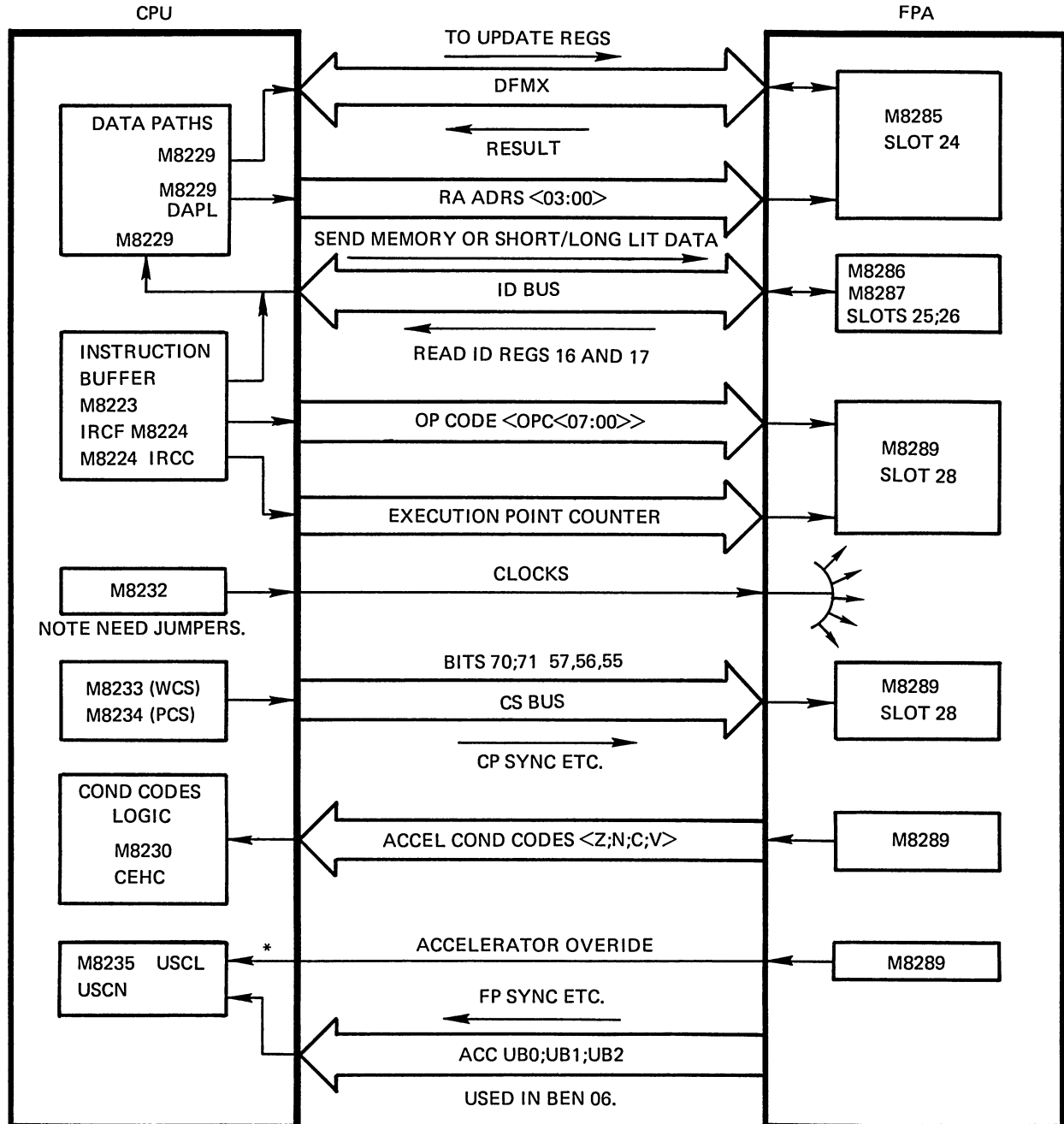
1. Power Supply No. 1 to be installed.

Example viewed from rear



2. Insert clock jumper on the M8232 clock module.

FPA/CPU Interface



IMPORTANT

- * THIS SIGNAL ACCELERATOR OVERRIDE SET UPC BIT 12 TO PUT CPU INTO WCS.

TK-0983

Microstates for FPA

MULF3 R1, R2, R3
HALT

CPU upc	FPA apc (simultaneous microwords)
0062	0180
0004	01FE
1204*	012E *VAX enters WCS
12D7	012E
124C	00CB
010D	010D
124C	011D
124D	00A7
1245	00A3
008F**	0180 **VAX re-enters PCS
039F	0180
00FF	0180

MULF (R1), R2, R3
HALT

CPU upc	FPA apc
0062	0180
0008	01F2
0094	0132
1204	012E
1204	012E
1204	012E
1204	012E
1204	012E
1204	012E
1204	012E
1204	012E
1204	012E
12D7	012E
124C	00CB
124C	010D
124C	011D
124D	00A7
1245	00A3
008F	0180
039F	0180
00FF	0180

FPA Floating-Point Multiply Instruction

Floating-Point Multiply

53525145
00000000

200/ MULF3 R1, R2, R3
203/ HALT

R1/00004080

R2/00004080

PCS	FPA/PCS	WCS	Comments from FPA/PCS
0062	0180		<p>uword 0A3, C008, 2C6E, B3A8 FPA Instruction Wait State assume register to register addressing wait here for valid FPA Opcode Take data from general register copies: BUS A gets R[SP1], BUS B gets R[SP2] Load AR1, LA, SA, MC1, MP0 from BUS A Load BR1, LB, SB, MP1, MC1 from BUS B (Take the exponent difference) (look up first multiplier nibble)</p>
0004	01FE		<p>uword 1FE, 8008, 04FF, B18A Float R, X First operand was a register, which was loaded in the previous state. Get R[PRN] and load it into LB, SB, BR1, MP1, and MC1 in case the second operand is a register. Take the exponent difference.</p>
	012E	1204*	<p>uword 12E, 65D3, 920C, 3C06 MULF X,R All operands are loaded. Issue the Continue signal to the multiplier, so it will multiply the fractions. Add the exponents. Go to the multiply execution flows. Branching on presence of zeroes/reserved operands. *comments for WCS, after FPA/PCS</p>

Floating-Point Accelerator

PCS	FPA/PCS	WCS	Comments from FPA/PCS
	00CB	124C	uword 0CB, 849A, 420C, 3C06 No zero exponent. Remove the excess-80 from the sum of the exponents. Now go to the separate execution flows.
	010D	124C	uword 10D, 8E88, 0000, 6C06 MULF The multiplications of the fractions is done. Move the fraction product from the SALU to the NR. Go to the normalization flows.
	011D	124C	uword 11D, 5248, A24C, 5C6F MULF is done Normalize and round the fraction and adjust the exponent accordingly. Output the result from the NSHF (fraction). The exponent processor, and the sign processor. Flag it with FPSYNC and go to store flows.
	00A7	124D	uword 0A7, 5188, C03C, 586F Float, CPSYNC The CPU took the result in the previous state. Initialize the accelerator and go to wait for another accelerator instruction.
	00A3	1245	0A3 is FPA IRD state with 180 contained within the next address field.
008F	0180		

Comments for the ROM Status in WCS

```

1204  uword 1204, 001F, 2640, 1580, BEF8, 0000, 12D3
      Entry status: 1)  MULD/PROD in GPR<SP1>
                    2)  MULR operand in D-register
                    3)  MULD/PROD operand in LA-register
      Send MULR operand in D-register to accelerator.
      2's complement of the MULR operand

12D7  uword 12D7, 0000, 067C, 01D8, F800, 0000, 124C
      Not MULL, go to Floating Routine
      Get FPA result if ready. Merge into single precision
      DST/SP1 RTN

124C  uword 124C, 0000, 067C, 01D8, F8000, 0000, 124C
      Accept FPA result if ready

124C

124C

124D  uword 124D, 4001, 263C, 0180, F8C5, 5800, 1245
      Got result and no exception. Result to destination
      register/IBUF01<R>. Load FPA condition codes into PSL.

1245  uword 1245, F80C, 003B, 01F1, F857, 139B, 6000
      Finished instruction.
      IRD

```

FPA ROM Word

ROM WORD INPUT=C0082C6EB3A8

OPLD=8,LD ALL RR	LD RR=0,HOLD RR	SGNC=5,LD SGN
FADC=3,LD AR1,BR1	BSC=B, R	SCR=2,SP1,SP2
NOR REG=3,NOP	MISC CONT=6, IR0	WAIT=0,NOP
EAC=C,LA<BUSA,LB<BUSB	MCTL=0, NOP	FPSYNC=1
EALC=0, SEL A	BMXC=0,NK	AMXC=2,PR
BEN=0,NOP	NAD=180	

ROM WORD INPUT=800804FFB18A

OPLD=A,LD INT MUL	LD RR=0,HOLD RR	SGNC=4,LD SB
FADC=1,LA BR1	BSC=B, R	SCR=3,PRN+1,PRN
NOR REG=3,NOP	MISC CONT=7, IR1	WAIT=1,WAIT FOR SYNC
EAC=4,LB < BUSB	MCTL=0, NOP	FPSYNC=0
EALC=0, SEL A	BMXC=0,NK	AMXC=2,PR
BEN=0,NOP	NAD=100	

ROM WORD INPUT=65D3920C3C06

OPLD=6,NOP CLEAR	LD RR=0,HOLD RR	SGNC=0,NOP
FADC=C,AR OUT	BSC=3,INT HI	SCR=0,CPU
NOR REG=3,NOP	MISC CONT=0, NOP	WAIT=0,NOP
EAC=2,LD PR	MCTL=1, MULT	FPSYNC=0
EALC=2, A+B	BMXC=3, LB	AMXC=0,LA
BEN=5	NAD=0CH	

ROM WORD INPUT=849A420C3C06

OPLD=6,NOP CLEAR	LD RR=0,HOLD RR	SGNC=0,NOP
FADC=C,AR OUT	BSC=3,INT HI	SCR=0,CPU
NOR REG=3,NOP	MISC CONT=0, NOP	WAIT=0,NOP
EAC=2,LD PR	MCTL=0, NOP	FPSYNC=0
EALC=1, A-B	BMXC=2, #80	AMXC=2,PR
BEN=1	NAD=109	

FPA ROM Word (Cont)

ROM WORD INPUT=8E8800006C06

OPLD=6,NOP CLEAR	LD RR=0,HOLD RR	SGNC=0,NOP
FADC=C,AR OUT	BSC=6,PQ	SCR=0,CPU
NOR REG=0,BUSA,BUSH	MISC CONT=0, NOP	WAIT=0,NOP
EAC=0,NOP	MCTL=0, NOP	FPSYNC=0
EALC=0, SEL A	BMXC=0,NK	AMXC=2,PR
BEN=0,NOP	NAD=11D	

ROM WORD INPUT=5248A24C5C6F

OPLD=F,INITIALIZE	LD RR=0,HOLD RR	SGNC=3,A<RESULT
FADC=C,AR OUT	BSC=5,NOR HI	SCR=0,CPU
NOR REG=3,NOP	MISC CONT=4, CC	WAIT=0,NOP
EAC=2,LD PR	MCTL=0, NOP	FPSYNC=1
EALC=2, A+B	BMXC=0,NK	AMXC=2,PR
BEN=4	NAD=0A4	

ROM WORD INPUT=5188C03C586F

OPLD=F,INITIALIZE	LD RR=0,HOLD PR	SGNC=3,A<RESULT
FADC=8,LD AR,BR	BSC=5,NOR HI	SCR=0,CPU
NOR REG=3,NOP	MISC CONT=3, RR<0	WAIT=0,NOP
EAC=0,NOP	MCTL=0, NOP	FPSYNC=0
EALC=3, K,3FF	BMXC=0,NK	AMXC=2,PR
BEN=0,NOP	NAD=0A3	

Example of FPA Multiply Algorithm

	MULTIPLICAND	278A	
	MULTIPLIER	<u>0F3B</u>	
Line 1		6E	ROM O/P B x A
2		58	ROM O/P B x 8
3		4D	ROM O/P B x 7
4		16	ROM O/P B x 2
5		1A2EE	Partial product(1+2+3+4)
6		1	Stored carry from PALU
7		1A2EE	AALU(5+ accumulated contents)
8		1E0	ROM O/P 3 x A
9		18	ROM O/P 3 x 8
10		15	ROM O/P 3 x 7
11		06	ROM O/P 3 x 2
12		0779E0	Partial product(6+8+9+10+11)
13		081BCE	AALU (7 + 12)
14		1→1	AALU stored carries
15		9600	ROM O/O F x A
16		78	ROM O/P F x 8
17		69	ROM O/P F x 7
18		1E	ROM O/P F x 2
19		14016	Partial product (15+16+17+18)
20		111	Stored carries from PALU
21		14922CE	AALU
22		1	Store carry AALU
23	Result =	25A32CE	Final add SALU(20+21+22)

The above shows a multiplication using the stored carry technique. In this example, we have a pipeline of adders, two of which implement stored carry. Both the pipeline and stored carry increase speed substantially. Stored carries in the above example are on nibble boundaries which can cause problems. For example, on lines 20, 21, and 22, we have the inputs for the final add. Certain conditions could cause stored carries from the PALU and the AALU to coincide, hence giving incorrect results. For this reason, the FPA offsets the stored carries produced by the PALU by one-half nibble. Without this offset, the final add would take two additions.

EXERCISES

The following is a series of floating-point macroinstructions. Treat them as a continuing sequence of instructions. Fill in the required information. The instructor will review the entire floating-point exercise.

Macro Code	Value in Memory	Value in Dest. Register
MOVF #1, R0	_____	_____
MNEGF R0, R1	N/A	_____
MOVF #2.5, R2	_____	_____
ADDF3 #3.5, R2, R3	_____	_____
CLRL R4	N/A	_____
ADDD3 R3, R3, R5	N/A	_____
ADDF2 #-1, R0	_____	_____
ADDF3 #0.5, R0, R1	_____	_____
MOVF #0.25, R0	_____	_____
MOVF #0.5, R1	_____	_____
MNEGF #1.5, R2	_____	_____
SUBF3 R1, R2, R3	N/A	_____
SUBF3 R0, R1, R4	N/A	_____
SUBF3 R1, R0, R5	N/A	_____
SUBF2 #0.5, R1	_____	_____
MOVF #0.0000005, R0	_____	_____
MOVF #100000000, R1	_____	_____

Macro Code	Value in Memory	Value in Dest. Register
MULF3 #0.5, #0.5, R4	N/A	_____
MULF3 R0, R1, R5	N/A	_____
MULD3 #1.5, #1.5, R6	N/A	_____
MULF2 #1, R0	N/A	_____
MOVF #1, R0	N/A	_____
MOVF #2, R1	_____	_____
MOVF #0.5, R2	N/A	_____
DIVF3 R2, R1, R3	N/A	_____
DIVF3 R0, R1, R4	N/A	_____
DIVF2 #0.5, R2	N/A	_____
MOVF #^X1, R1	_____	_____

MODULE TEST

1. Convert 4.75 base (10) to its floating-point format, as it would appear in a CPU general purpose register.
 - a. 0000419B
 - b. 00004189
 - c. 00004198
 - d. 00008198
2. When the FPA operation result is ready to be transmitted to the CPU
 - a. CP Sync is asserted
 - b. Accelerator Override is asserted
 - c. FP Sync is asserted
 - d. FP Trap is asserted
3. The FPA result is transferred to the CPU D-register via the DF multiplexer bus. When the CPU has the data it asserts
 - a. FP Sync
 - b. CP Sync
 - c. Accelerator Override
 - d. CP Trap
4. Once the FPA has all required data
 - a. FP Sync is asserted
 - b. CP Sync is asserted
 - c. Accelerator Override is asserted
 - d. ACC Trap is asserted

For the following floating-point format number CCCDBECC, answer the following

5. The sign of the exponent is _____
6. The sign of the fraction is _____
7. Value of the exponent in base (10) is _____
8. Value of the fraction in base (10) is _____

MODULE TEST

9. The floating-point format 00004080 represents
- a. -1
 - b. +1
 - c. -.1
 - d. +.1
10. The microsequence needed in the FPA to perform the following instruction ADDF2 R1, R2 is (no traps or special cases for this example)

MS-780 MEMORY SYSTEM

INTRODUCTION

In this module you will learn about memory addressing, error correction and detection, memory registers, and backplane strapping. In lab you will be able to use the microdiagnostics for fault detection analysis.

OBJECTIVES

1. Given the following function:
 - a. Memory receives a READ DATA command
 - b. Memory receives a WRITE DATA command
 - c. Memory detects and corrects a single-bit error
 - d. Memory detects a multiple-bit error
 - e. Memory receives a READ/WRITE INTERNAL REGISTER commandtrace it on a functional block diagram of the memory system.
2. Given a read data word and the stored check bits, identify the error syndrome and the bit in error.

SAMPLE TEST ITEM

On the supplied copies of the memory system functional block diagram, draw the necessary data path needed to perform the following types of data transfer between a SBI nexus and memory nexus.

1. Memory receives a WRITE DATA command.

RESOURCES

1. MS-780 Memory System Technical Description
2. MS-780 Engineering Drawings
3. 11-780 Engineering Drawings

LECTURE OUTLINE

- I. Introduction
- II. Memory Address Space
- III. SBI Review
- IV. Configuration Registers
- V. Block Diagram Description
- VI. Memory Timing
- VII. Memory Flow
- VIII. Block Diagram Signal Flow
- IX. Backplane Strapping
- X. Memory Diagnostics
- XI. Memory Lab

BLANK

EXERCISES

1. Initialize the system.
2. Examine the memory configuration registers and the contents.
 - a. _____
 - b. _____
 - c. _____
3. How much memory does the system have?
4. What is the starting address of the memory?
5. Change the starting writable address by 256K bytes if MS 780C (16K chip), or by 64K bytes if MS780 (4K chip). For proof that the starting address was changed, a deposit to address 0 will create a MIC error.
6. Return memory address to 0.

Using the correct bits in the registers, answer the following questions.

- a. What type of memory device is used in the system, and what bits inform you of that fact and where?
- b. What is the status of memory initialization bits in the register?
- c. What is the status of the file?
- d. What is the condition of bit 22 in register A?

EXERCISES

Generation of Check Bits

Given a quadword, generate check bits as follows.

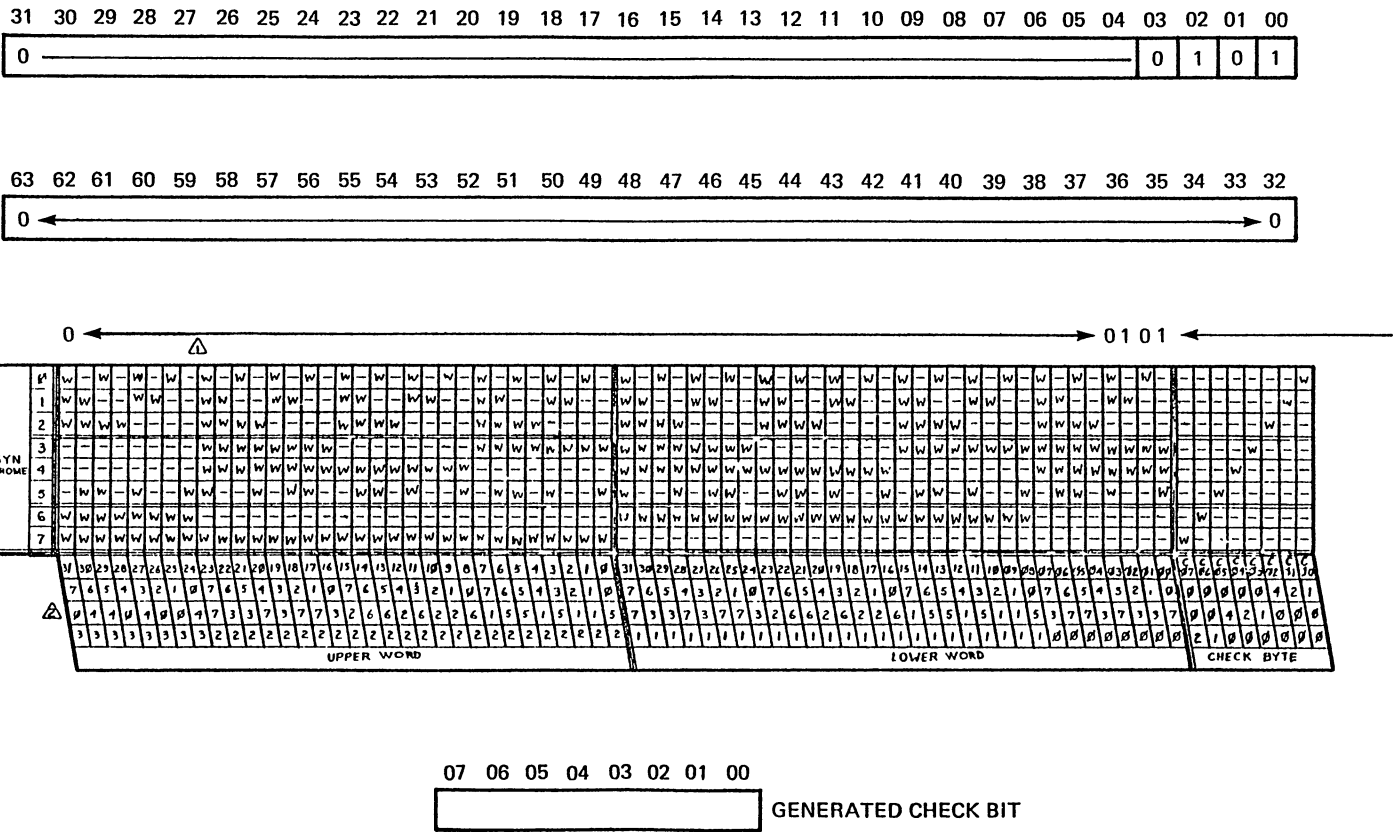
1. Examine given word.
2. Use chart on sheet 18 of 18 MDT print.
3. An all zero word will create check bit of [FF] (proof will follow later)
4. A nonzero word must be calculated by alternative methods. Below is one alternative method.
 - a. Place word in position. (Figure 1)
 - b. Match the 1s with the Ws in syndrome position 0.
 For bit check 0 - no match - generate 1 in bit check position 0. (Figure 2)
 - c. Match the 1s with the Ws in syndrome position 1.
 For bit check 1 - one match - an odd number generate a 0 in bit check position 1. (Figure 3)
 - d. Match the 1s with the Ws in syndrome position 2.
 For bit check 2 - no match - generate 1 in bit check position 2.
 - e. Match the 1s and Ws in syndrome position 3.
 For bit check 3 - even match - must be odd - so generate a 1 in bit check position 3. (Figure 5)
 - f. Match the 1s and Ws in syndrome position 4.
 For bit check 4 - even match - must be odd - so generate a 1 in bit check position 4 (Figure 6).

Using and repeating above steps will create final check bits of

11011101

(Figure 7)

GIVEN A QUADWORD, GENERATE CHECK BITS.



9-7

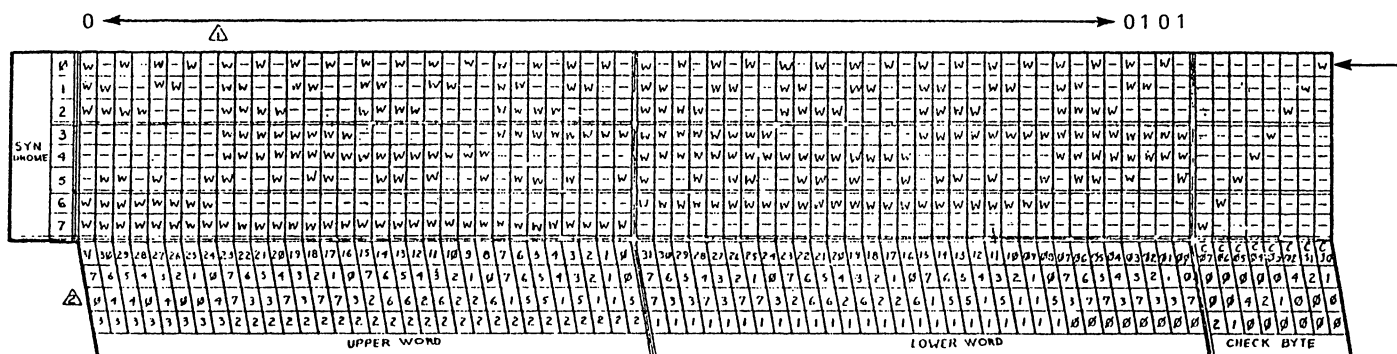
Figure 1

TK-0882

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
0																												0	1	0	1

63 62 61 60 59 58 57 56 55 54 53 52 51 50 49 48 47 46 45 44 43 42 41 40 39 38 37 36 35 34 33 32

0 ← ————— → 0



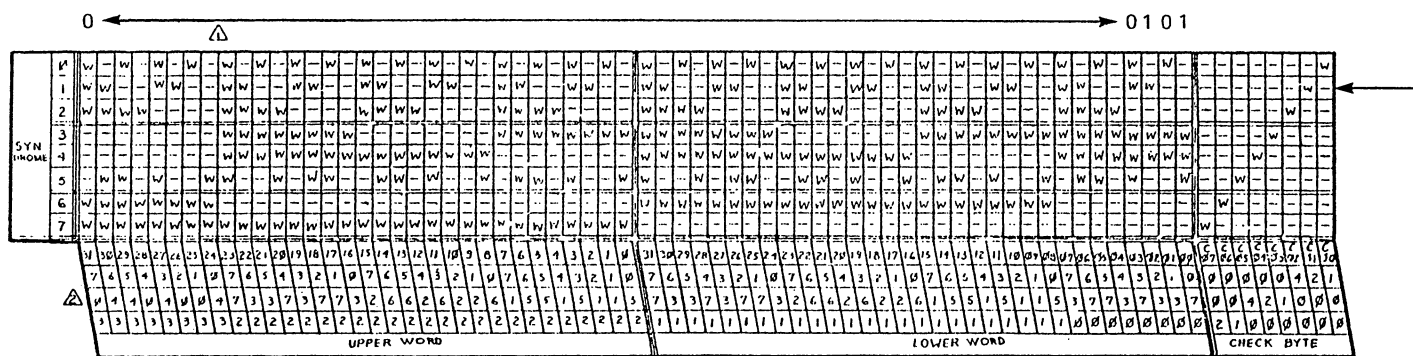
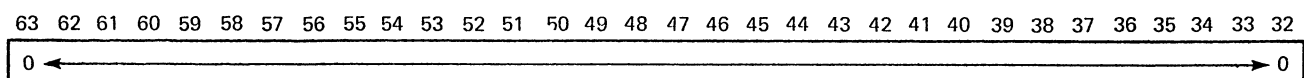
07	06	05	04	03	02	01	00
							1

GENERATED CHECK BIT

Figure 2

TK-4758

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
0																												0	1	0	1



07	06	05	04	03	02	01	00
						0	GENERATED CHECK BIT

Figure 3

TK-4759

GIVEN A QUADWORD, GENERATE CHECK BITS.

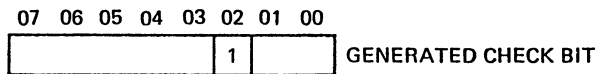
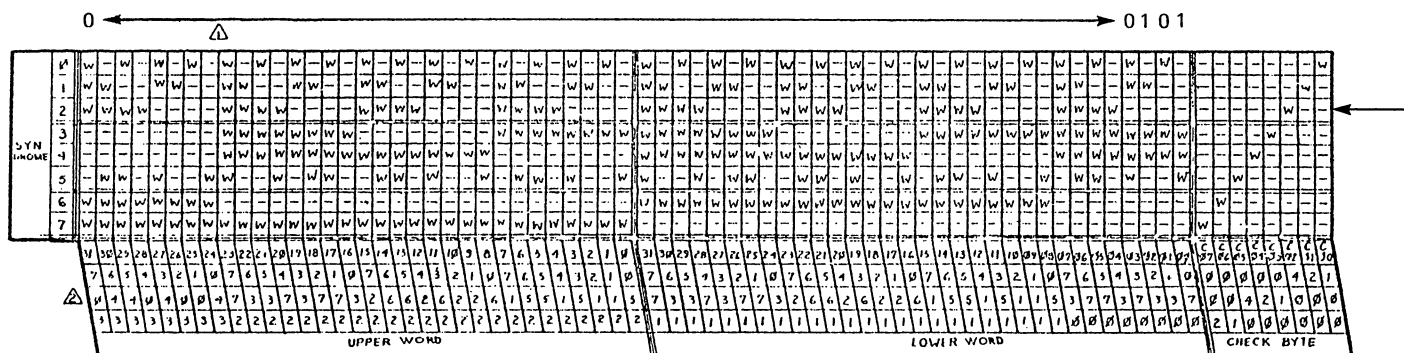
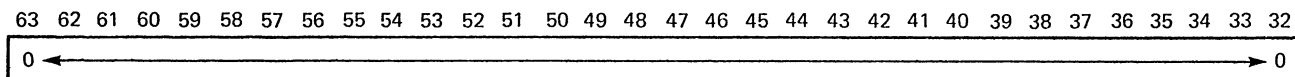
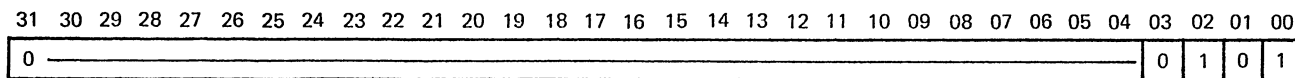
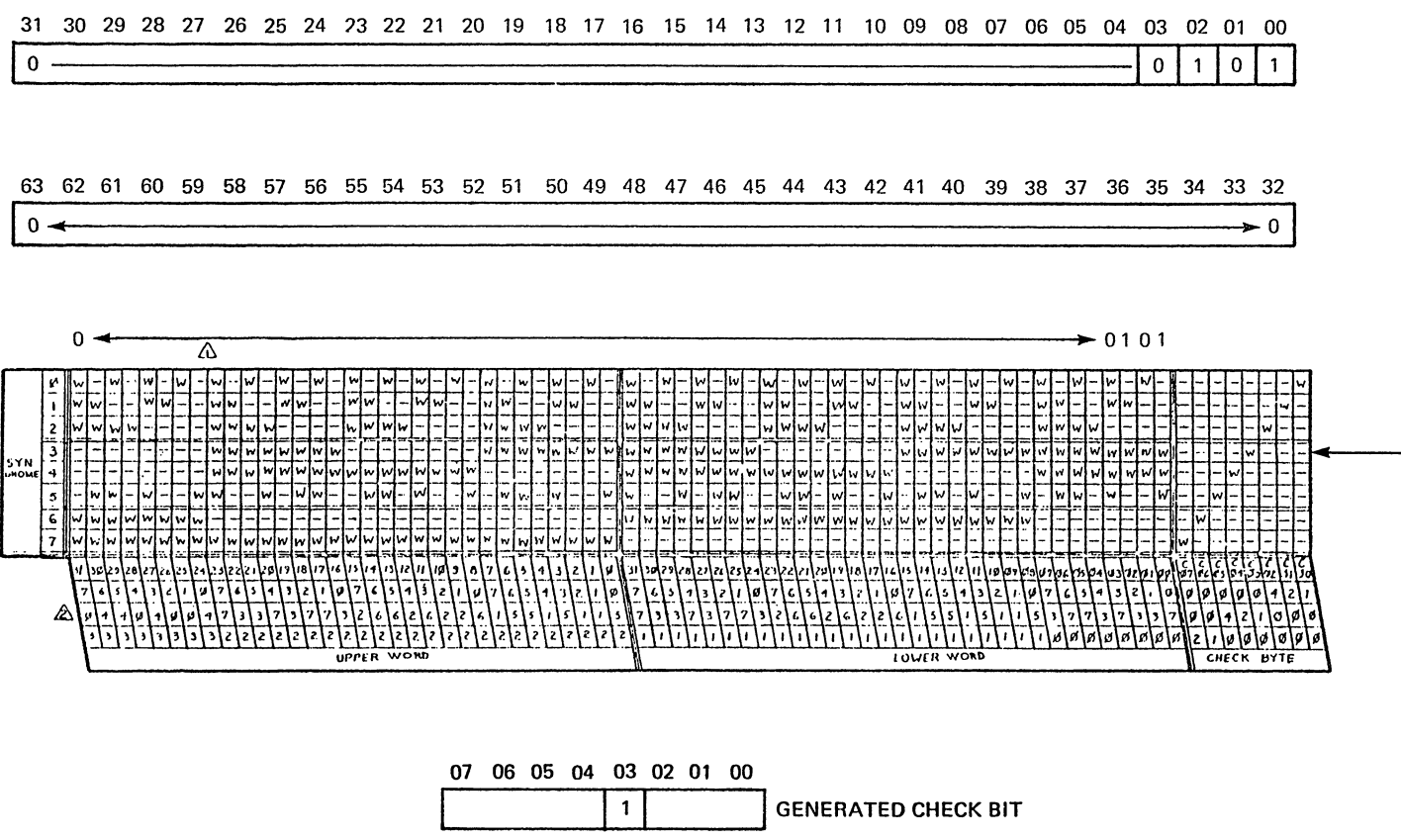


Figure 4

TK-4760

GIVEN A QUADWORD, GENERATE CHECK BITS.



9-11

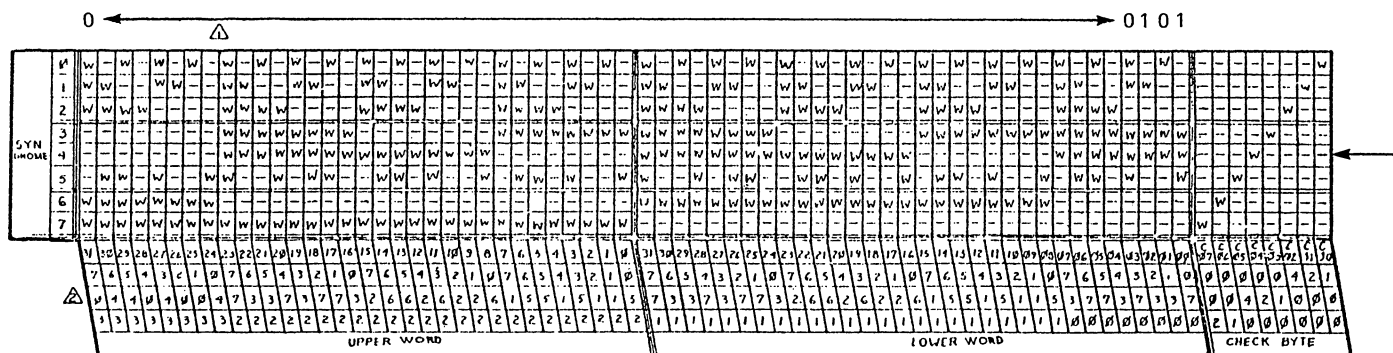
Figure 5

TK-4761

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
0																												0	1	0	1

63 62 61 60 59 58 57 56 55 54 53 52 51 50 49 48 47 46 45 44 43 42 41 40 39 38 37 36 35 34 33 32

0 ← → 0



07	06	05	04	03	02	01	00
			1				GENERATED CHECK BIT

Figure 6

GIVEN A QUADWORD, GENERATE CHECK BITS.

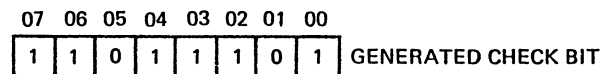
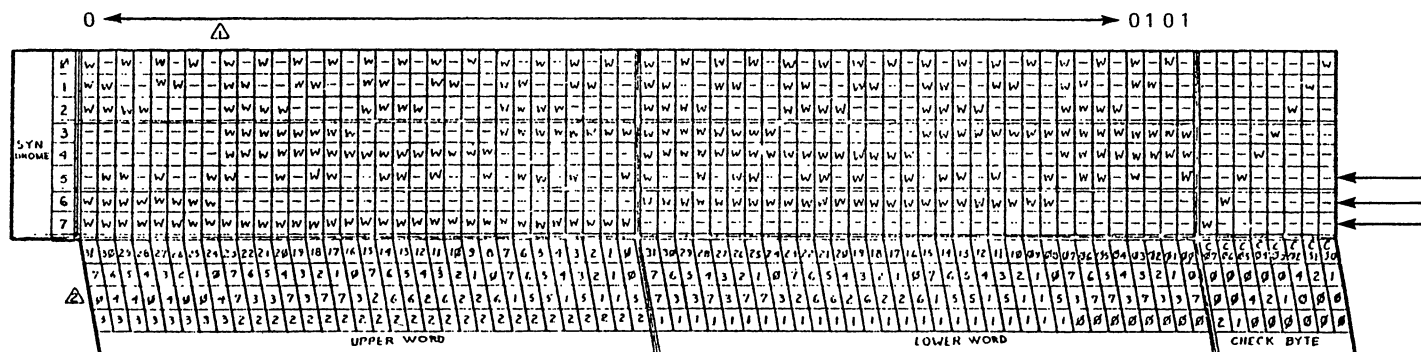
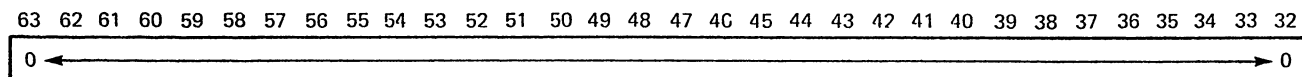
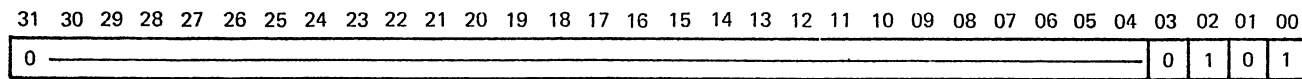
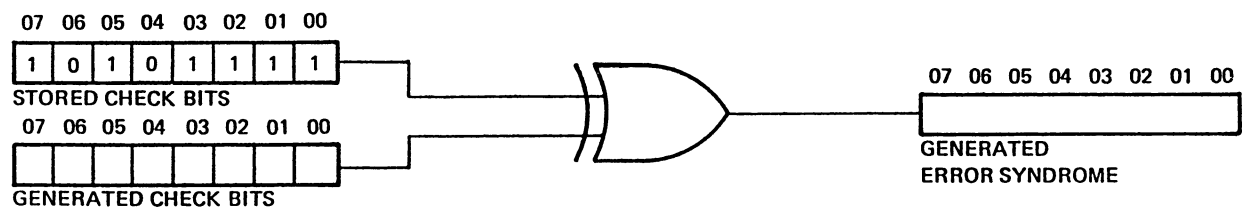
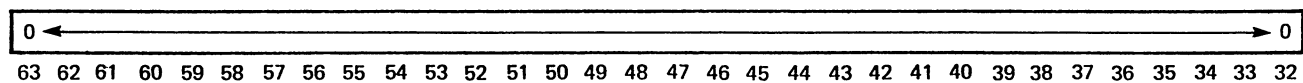


Figure 7

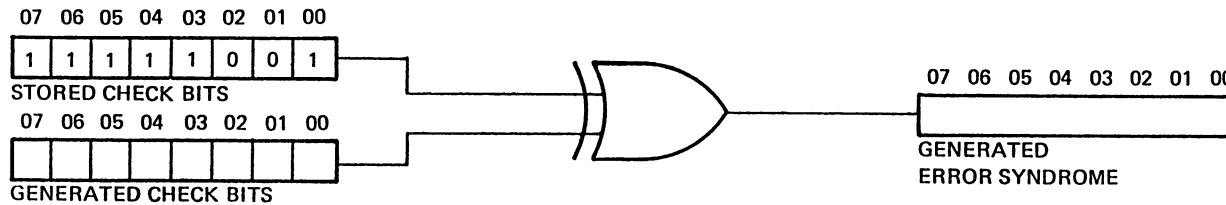
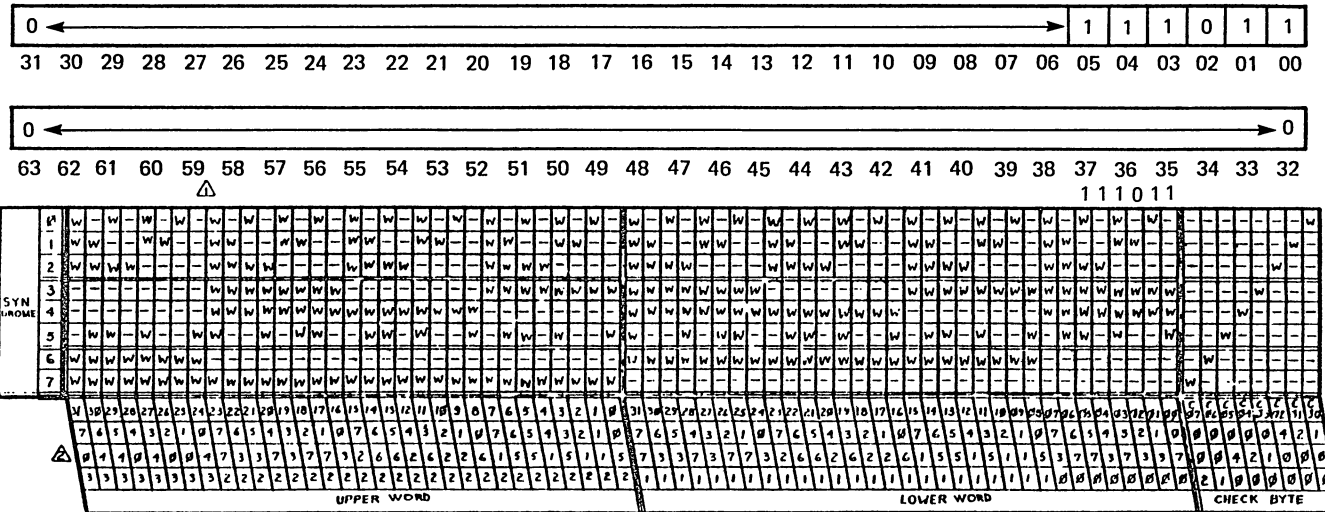
TK-4763

9-14



1. GENERATE CHECK BITS
2. GENERATE ERROR SYNDROME

GIVEN: A READ DATA WORD AND STORED CHECK BITS, DETERMINE SYNDROME,
BITS IN ERROR AND GENERATED CHECK BITS.



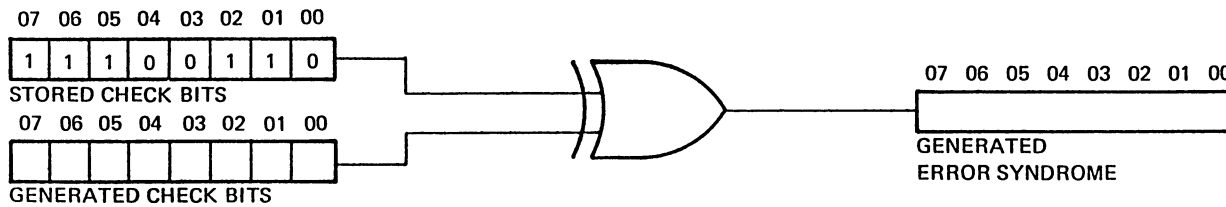
PERFORM

1. GENERATE CHECK BITS
2. GENERATE ERROR SYNDROME

3. BIT TO BE CORRECTED →
(IF ERROR IS CORRECTABLE)
- IF NONCORRECTING
CIRCLE NC.

9-16

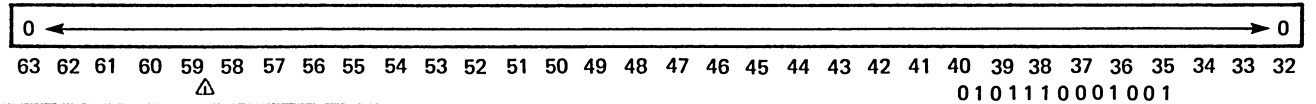
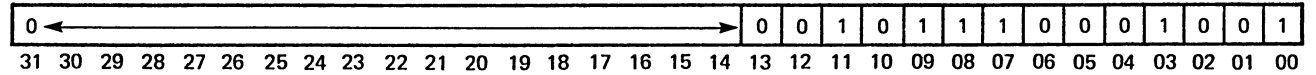
9-17



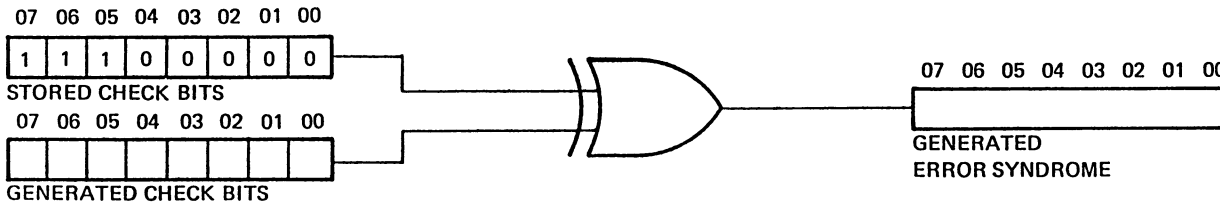
1. GENERATE CHECK BITS
2. GENERATE ERROR SYNDROME

ECC Worksheet D

GIVEN: A READ DATA WORD AND STORED CHECK BITS, DETERMINE SYNDROME,
BITS IN ERROR AND GENERATED CHECK BITS.



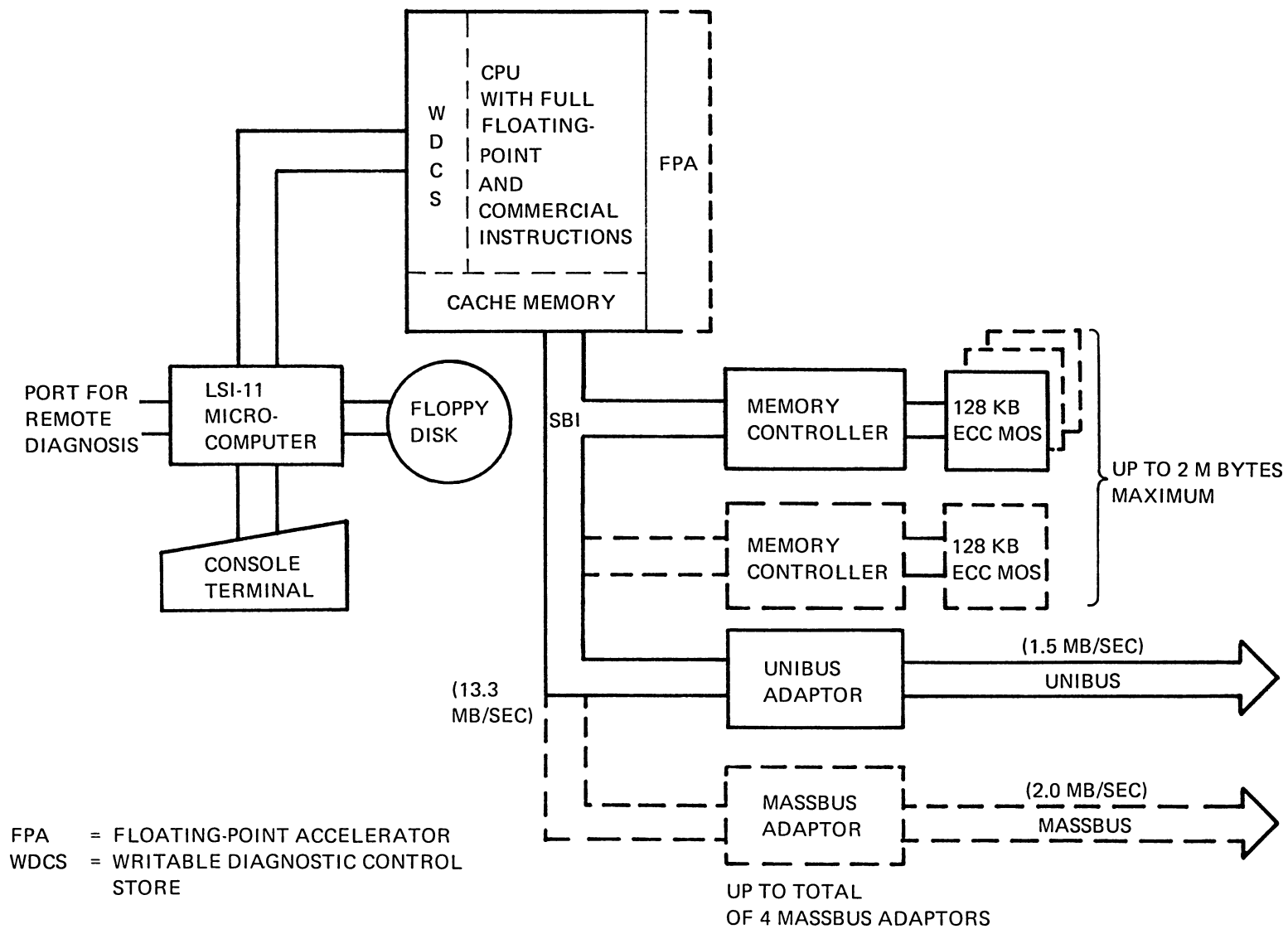
SYN- DROME	63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
1	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W
2	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W
3	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W
4	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W
5	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W
6	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W
7	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W



PERFORM

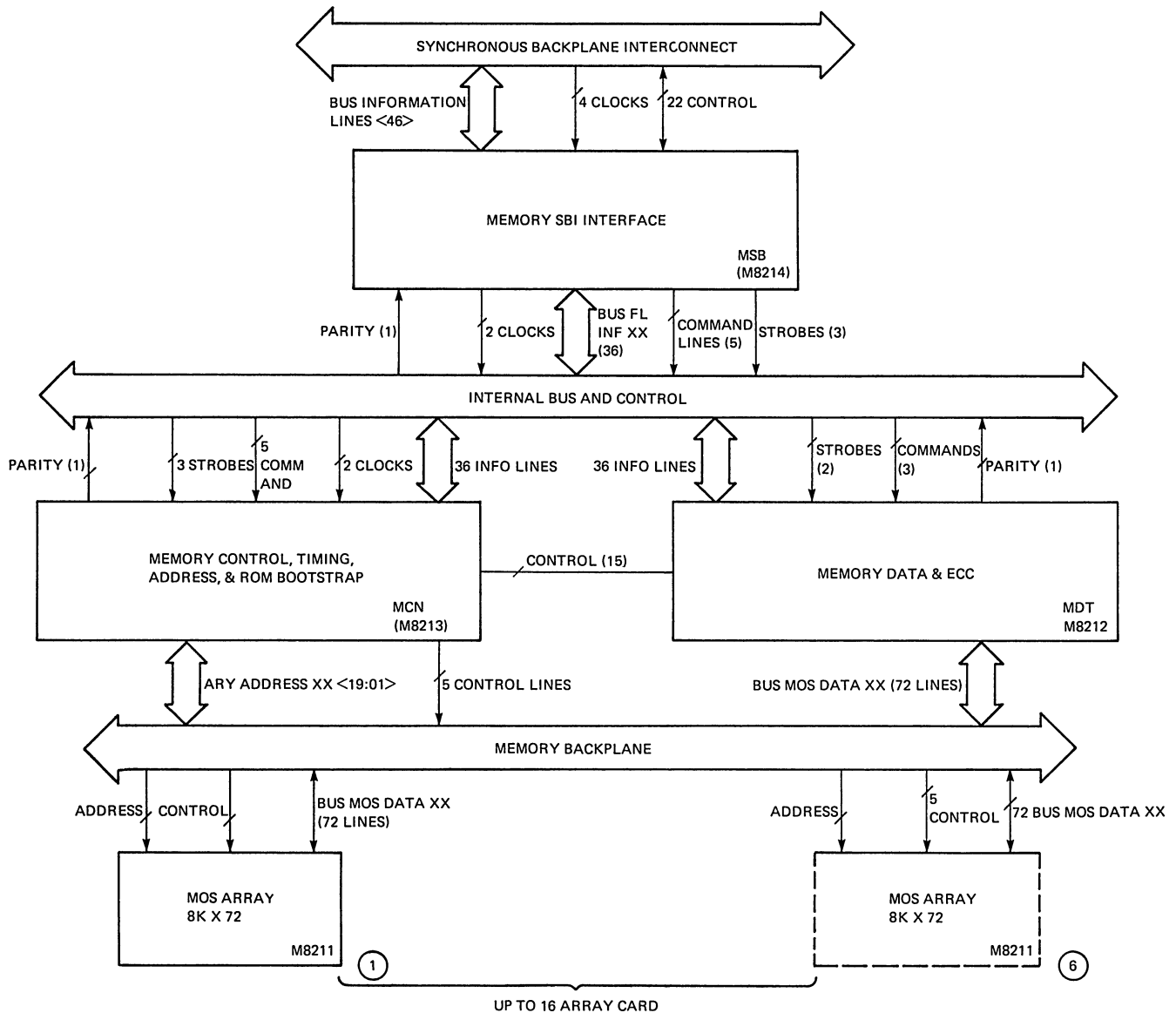
1. GENERATE CHECK BITS
2. GENERATE ERROR SYNDROME

3. BIT TO BE CORRECTED → _____
(IF ERROR IS CORRECTABLE)
- IF NONCORRECTING
CIRCLE NC. _____ NC



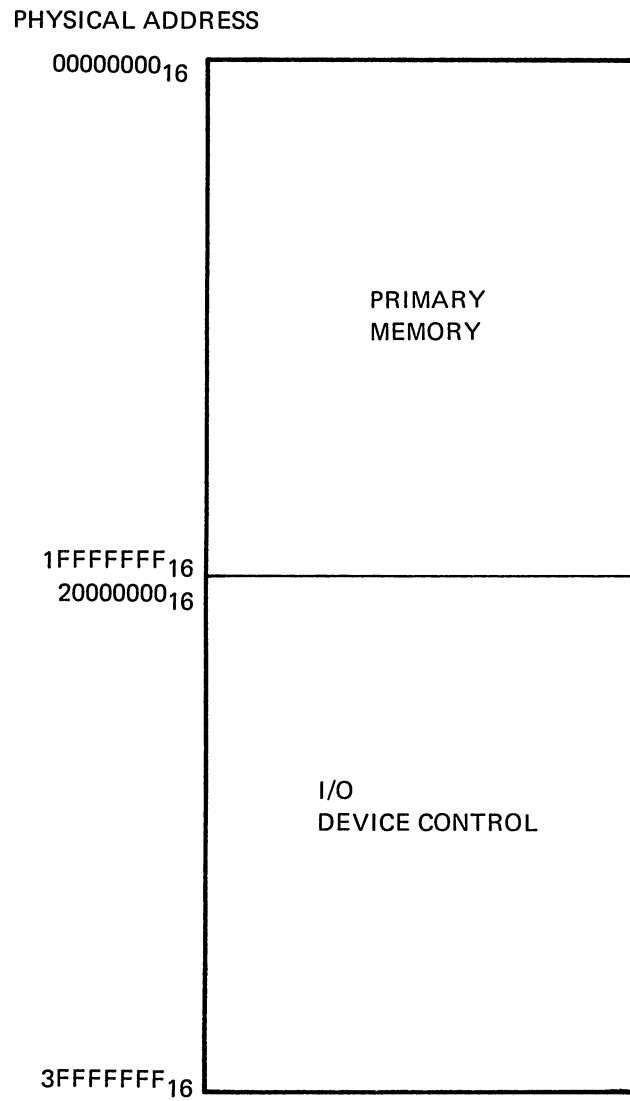
VAX-11/780 System

TK-4353



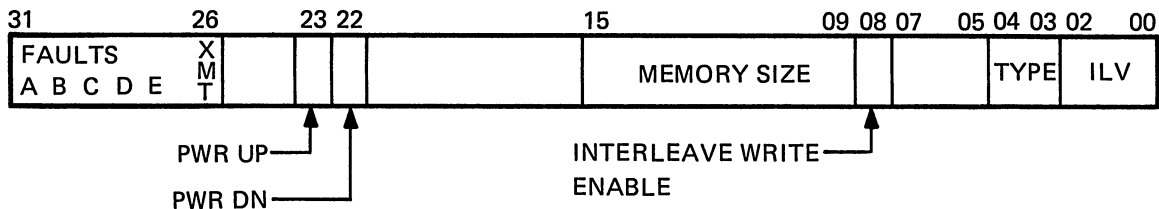
TK-4354

MS-780 Memory System Block Diagram

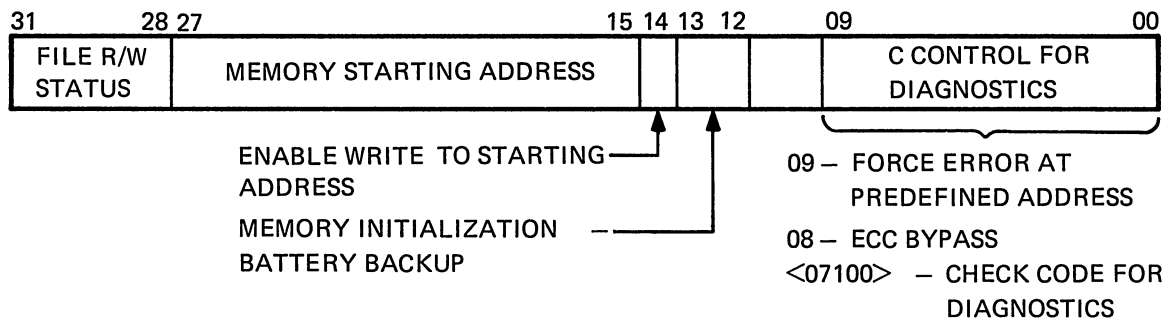
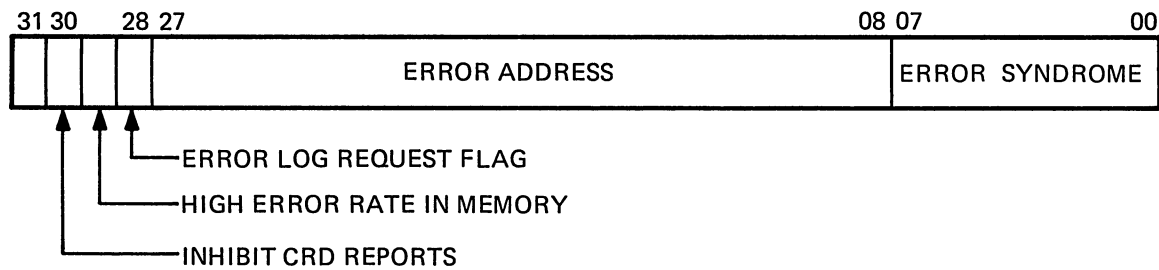


TK-0878

Physical Address Space

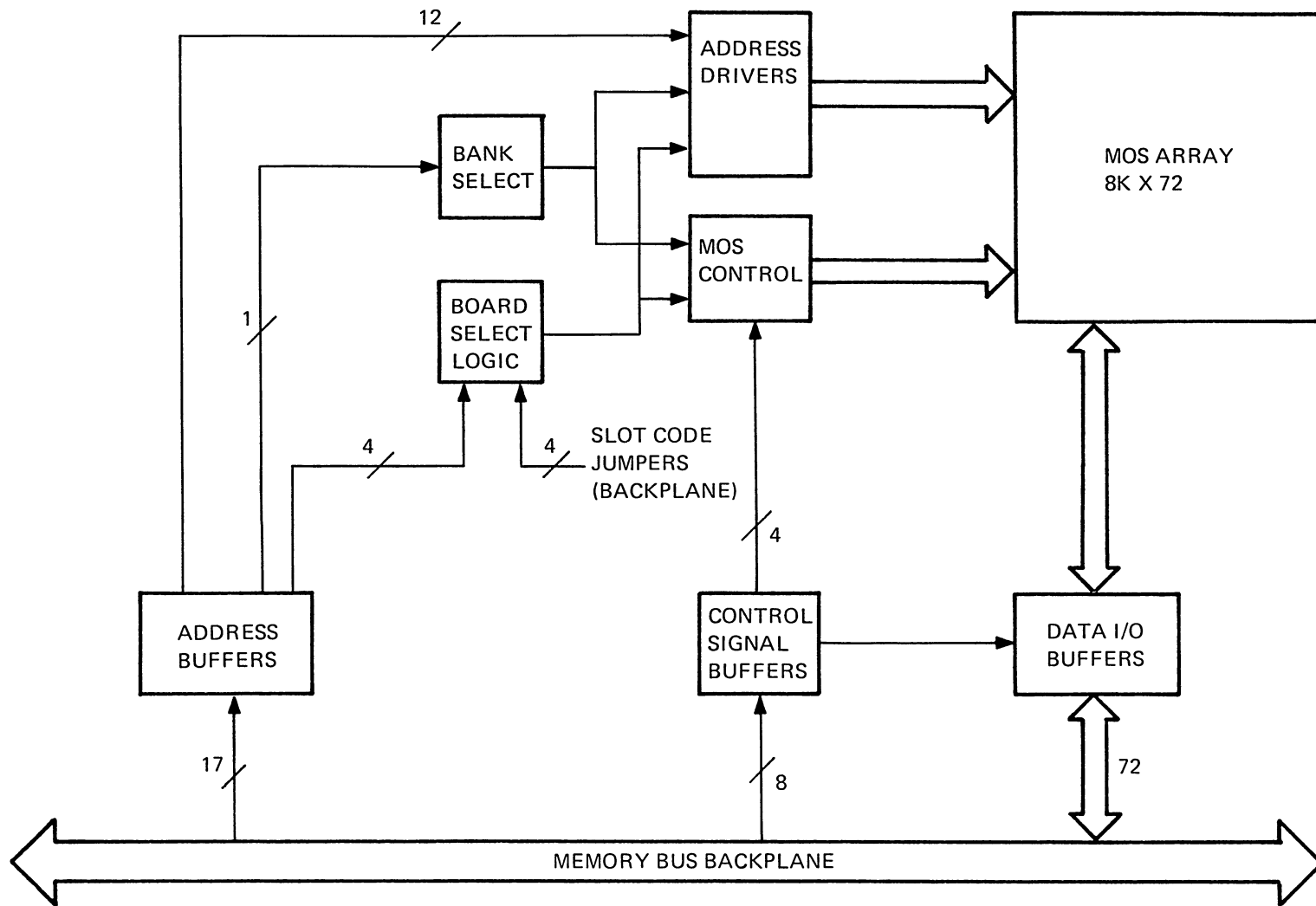
REGISTER A

- A – BUS PARITY ERROR
- B – WRITE DATA SEQUENCE FAULT
- C – (NOT USED IN MEMORY)
- D – ILK COMMAND SEQUENCE FAULT (ILK WRITE W/O ILD READ)
- E – MULTIPLE TRANSMITTERS ON THE BUS
- XMT – TRANSMITTER DURING CYCLE THAT CAUSED FAULT

REGISTER B**REGISTER C**

TK-3620

Memory Configuration Registers



TK-4349

Array Block Diagram

BLANK

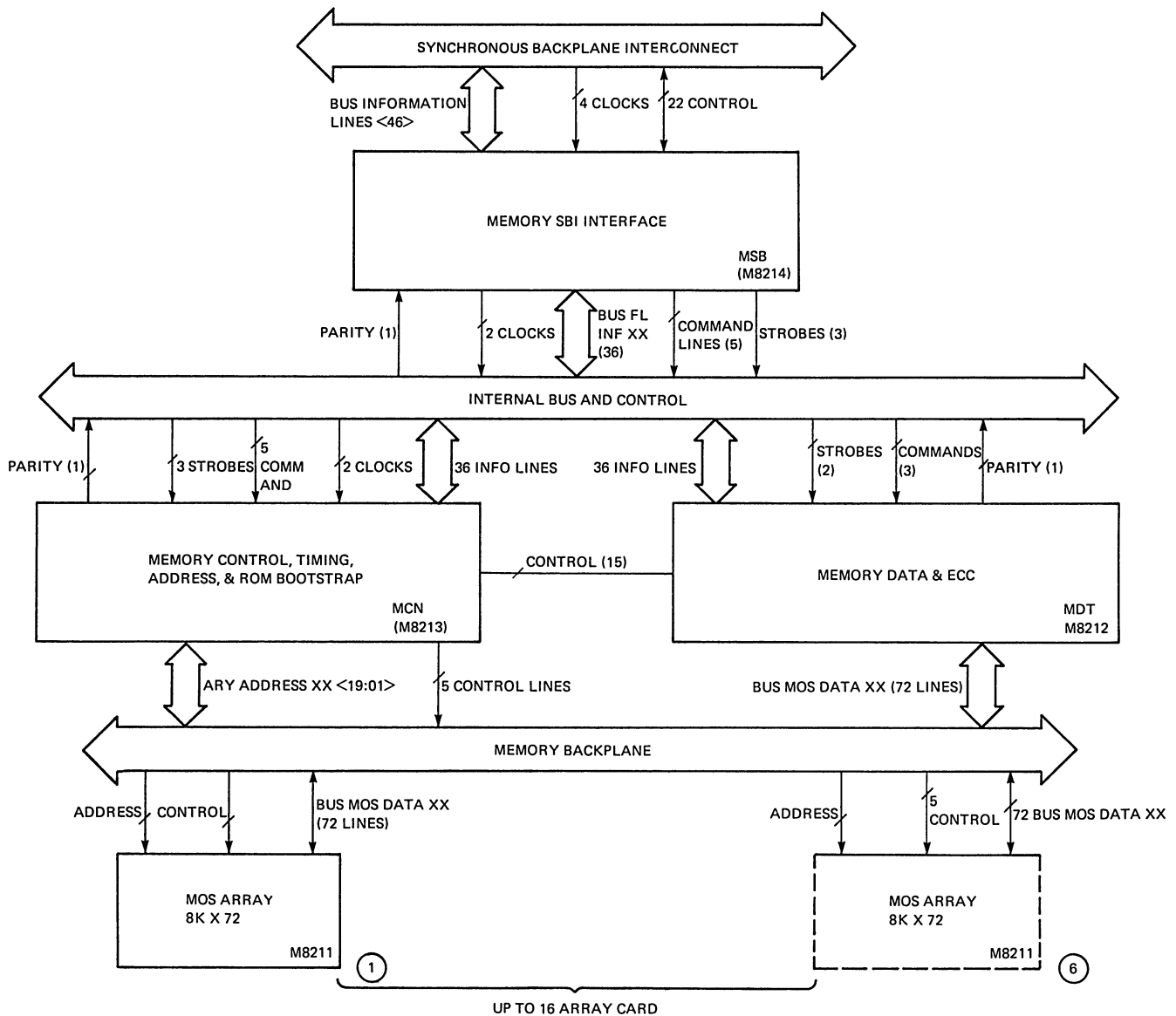
MODULE TEST

On the supplied copies of the memory system functional block diagram, draw the data path needed to perform the following types of data transfer between an SBI nexus and memory nexus.

1. Memory receives a WRITE DATA command.
2. CPU wants to read memory configuration register.
3. Given a read data and stored check bits, determine if an error syndrome is generated.

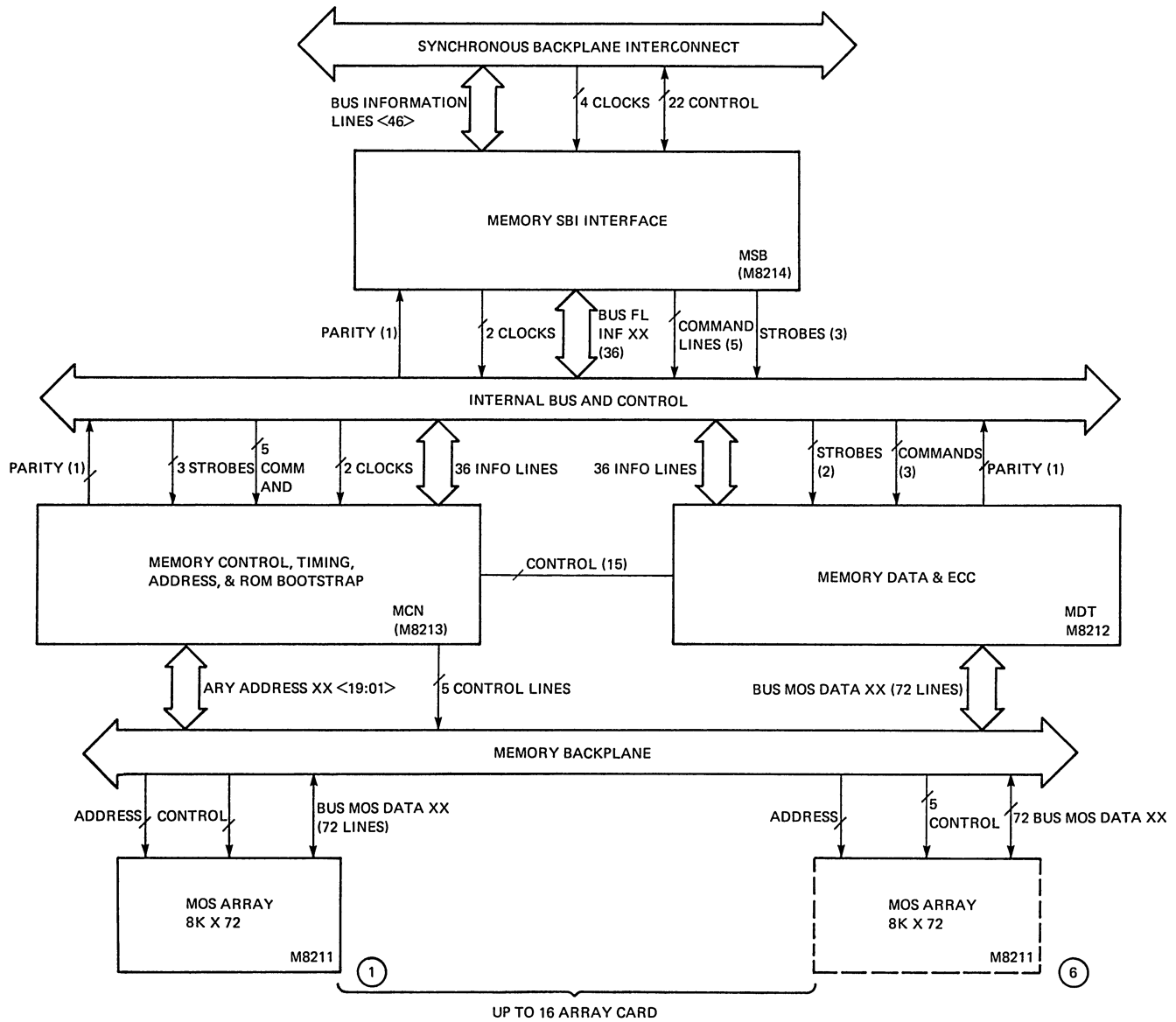
Read Data/ 00000000000000015
Store Check Bits/ DD

If the error syndrome is nonzero, what bit is in error?



TK-4354

MS-780 Memory System Block Diagram



TK-4354

MS-780 Memory System Block Diagram

BLANK

RH-780 MASSBUS ADAPTER

INTRODUCTION

The MASSBUS adapter (MBA) is the hardware interface between the synchronous backplane interconnect SBI and the high-speed MASSBUS storage devices. The MASSBUS is also the communication path linking the MASSBUS adapter to the mass storage device drives.

OBJECTIVES

Given the following function:

- External register read/write
- Internal register read/write
- Data transfer read/write to any MASSBUS device

trace it on a functional block diagram of the MASSBUS adapter.

SAMPLE TEST ITEM

On the supplied copy of the MASSBUS functional block diagram, draw the data path needed to perform this function:

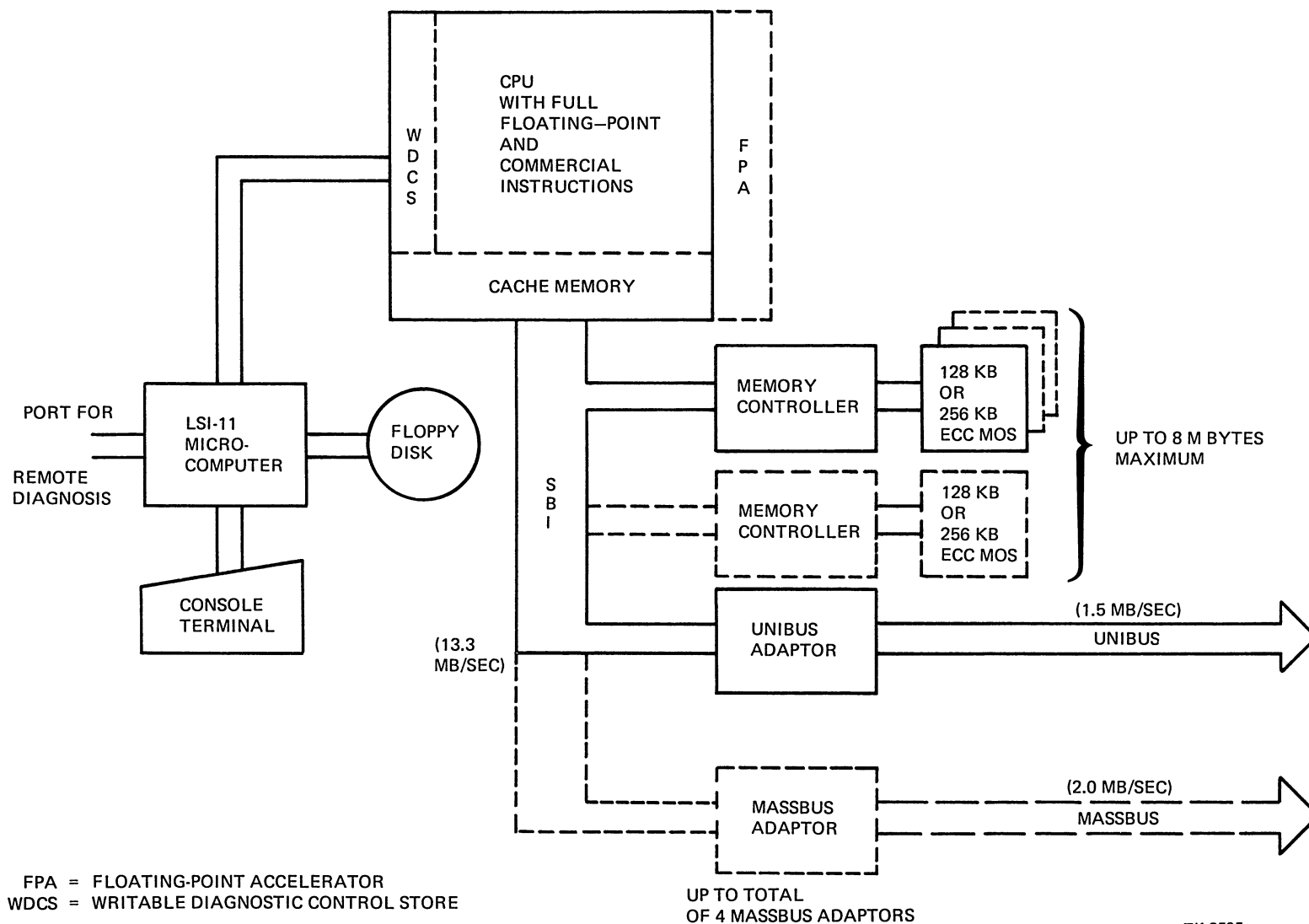
VAX-11/780 reads a register on a MASSBUS device.

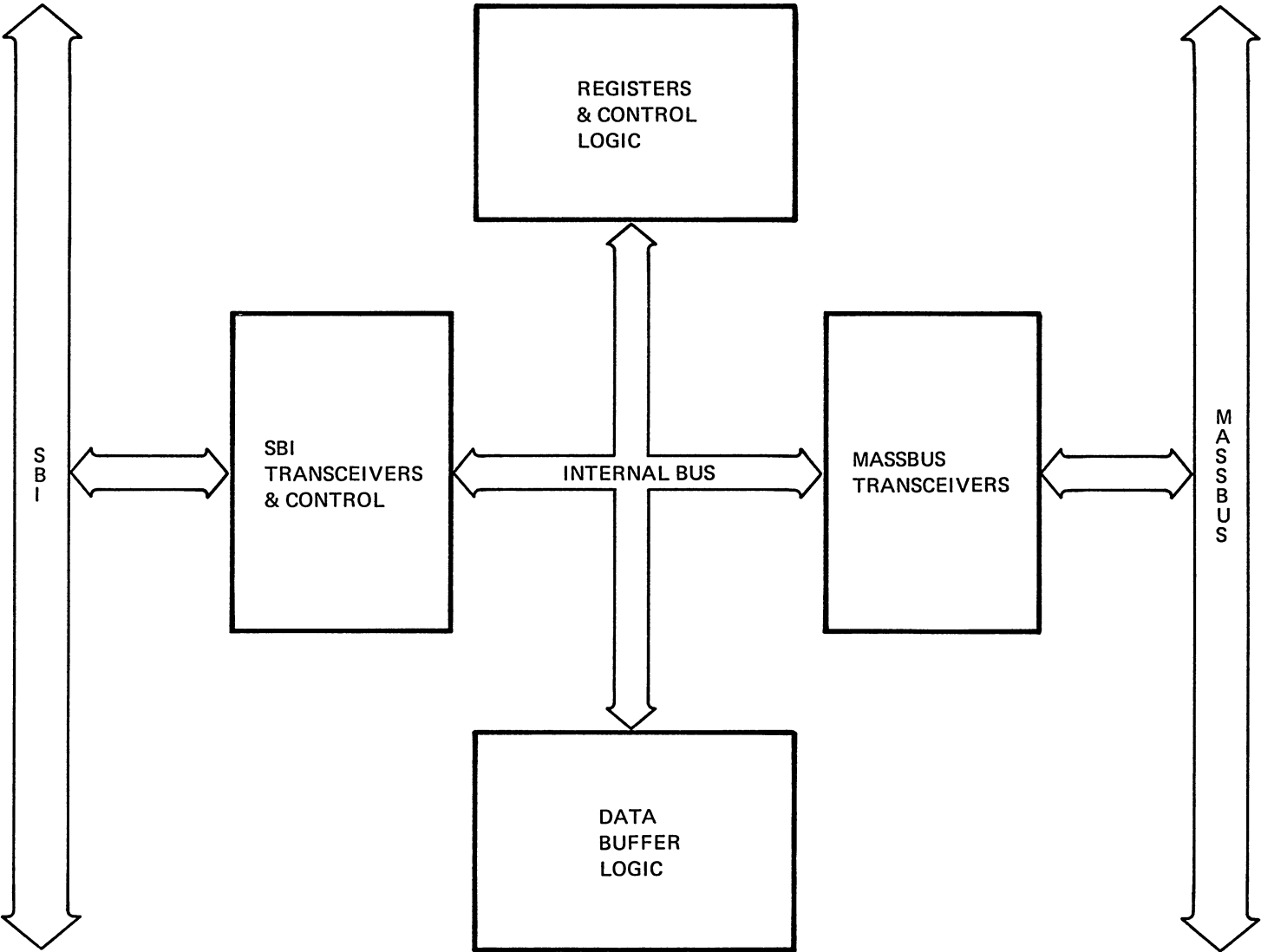
RESOURCES

1. RH-780 Massbus Adapter Technical Manual
2. RH-780 Engineering Drawings
3. VAX-11/780 Macrodiagnostics and Listings

LECTURE OUTLINE

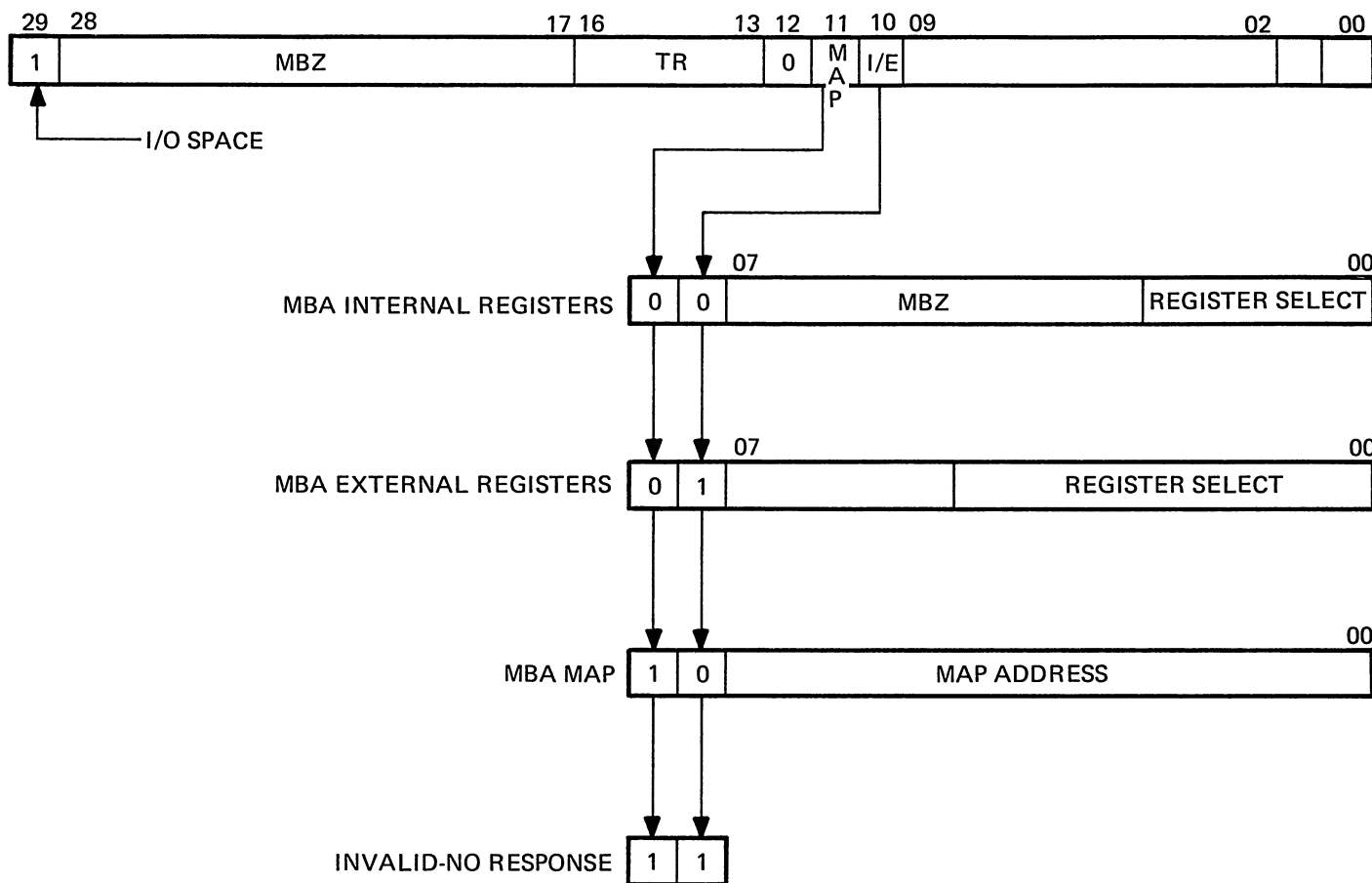
- I. Introduction
- II. MASSBUS Adapter Characteristics
- III. MASSBUS Adapter Overview
- IV. MASSBUS Adapter Diagnostic Overview
- V. Address Translation
- VI. MASSBUS Register Overview
- VII. Functional Block Diagram Description
- VIII. Backplane Jumpering
- IX. Lab Project





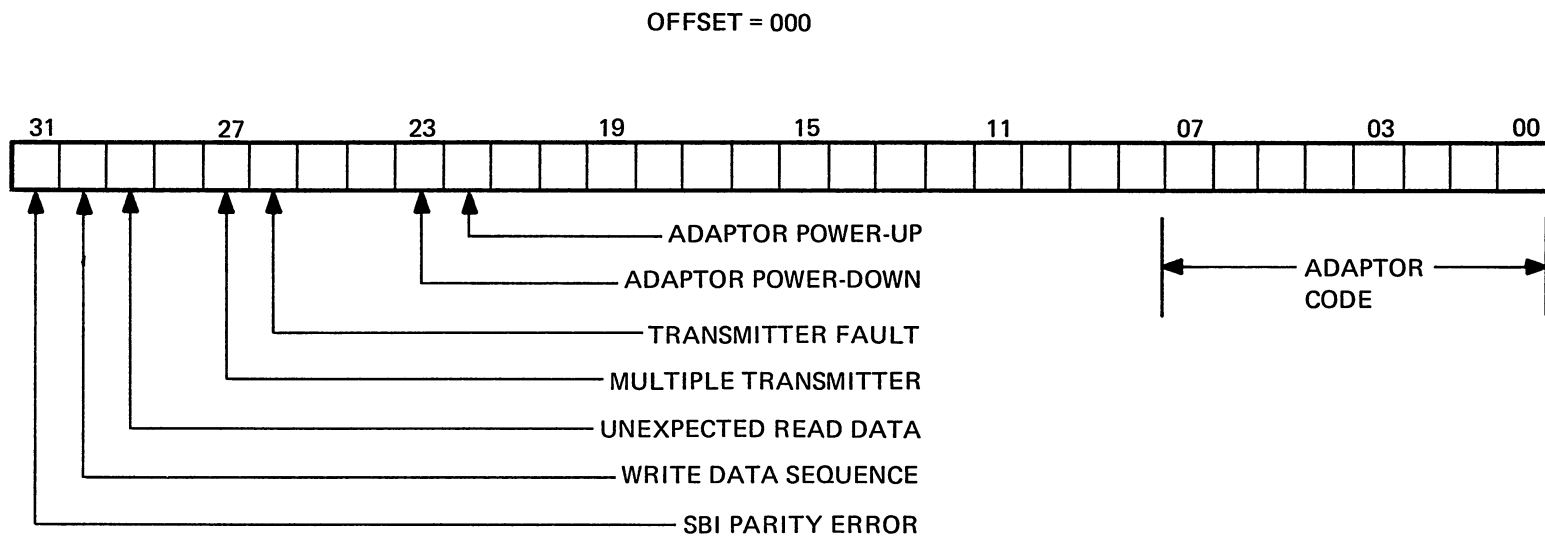
TK-1102

Simplified MBA Block Diagram



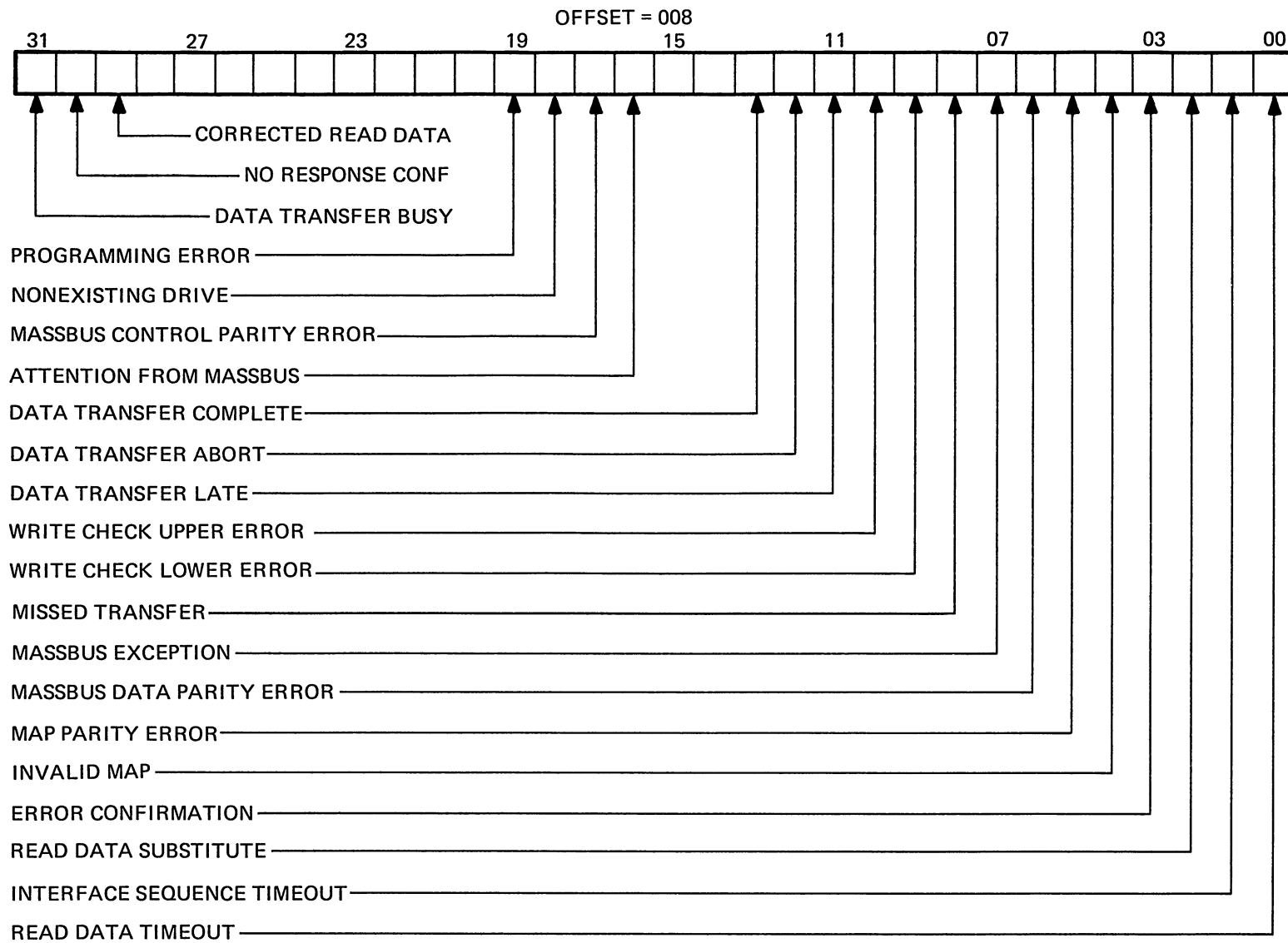
TK-1101

MASSBUS Adapter Addressing



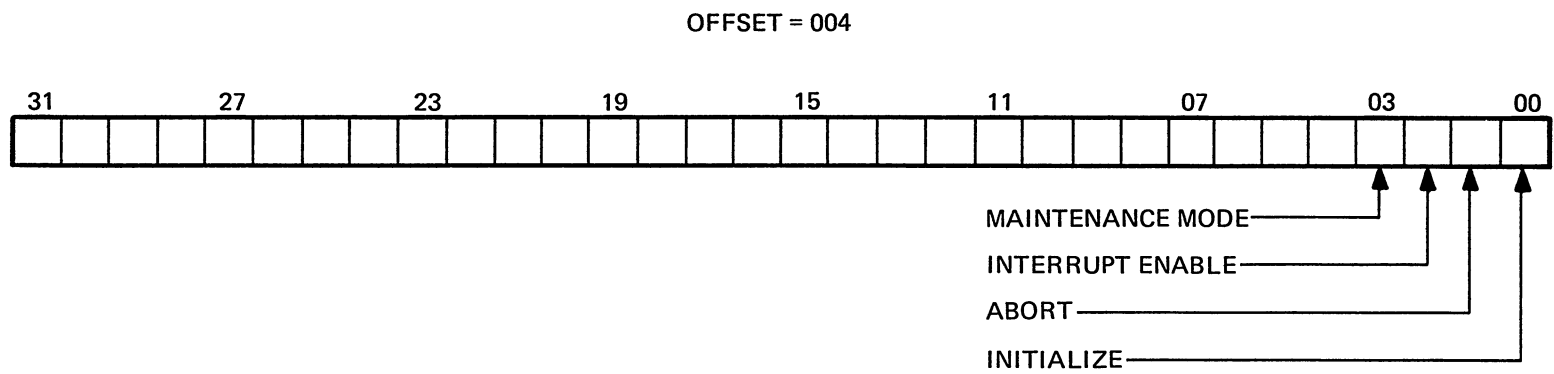
TK-1107

MBA Configuration Register



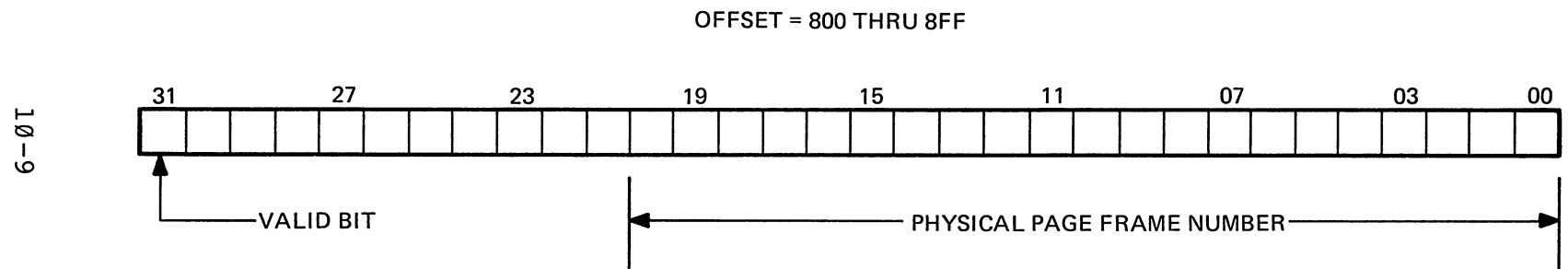
TK-1108

MBA Status Register



TK-1105

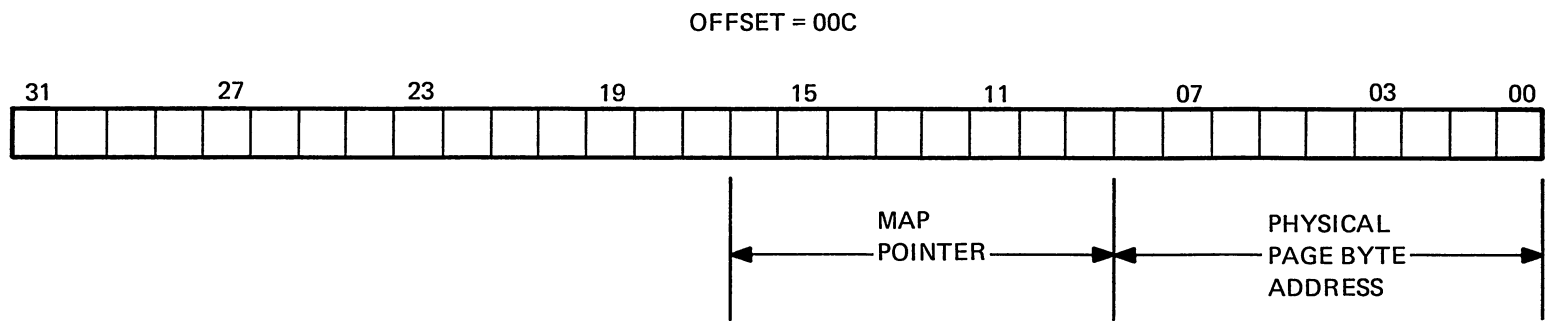
MBA Control Register



MBA Map Registers

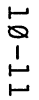
TK-1106

10-10



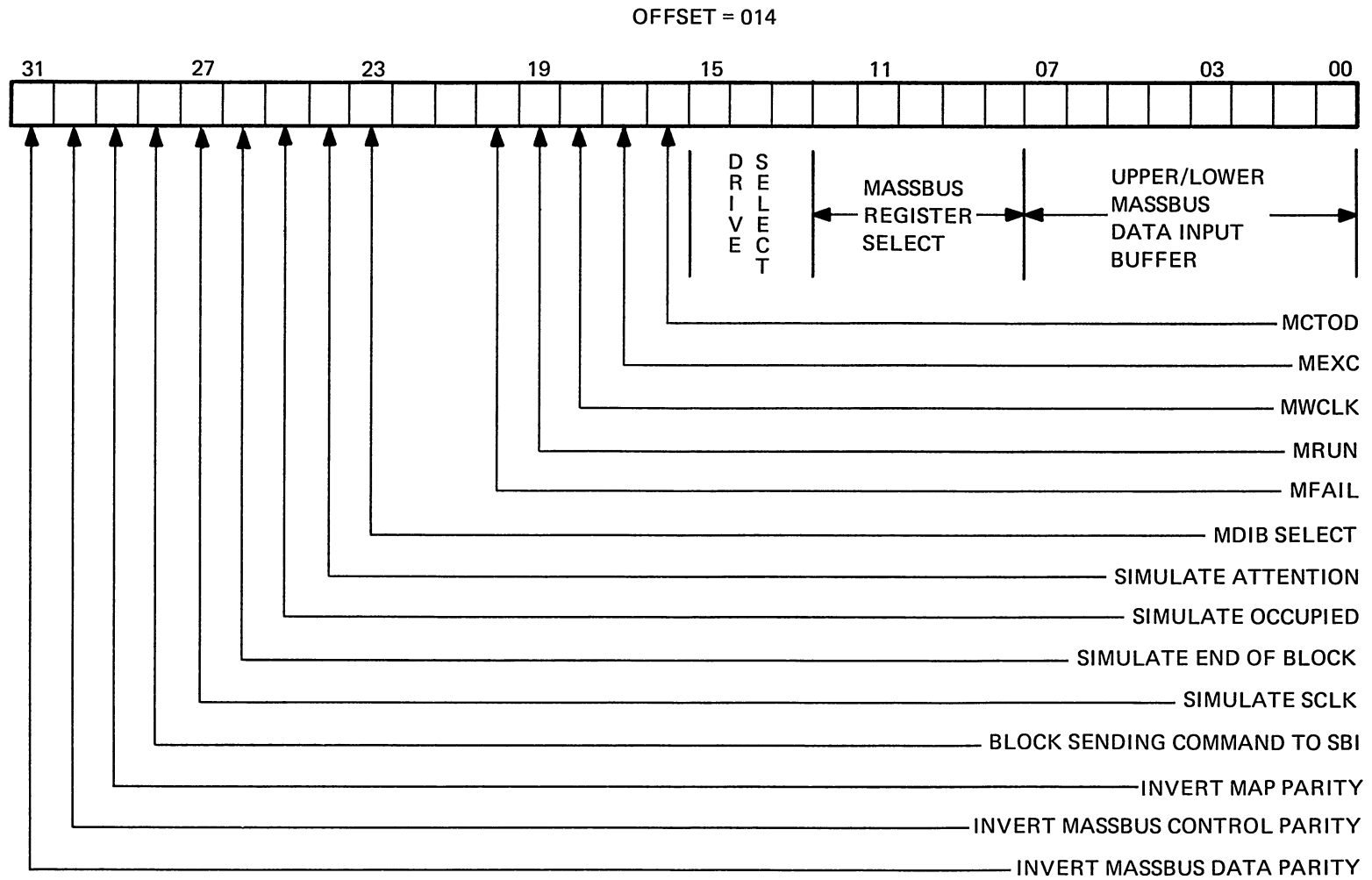
TK-1100

MBA Virtual Address Register



TK-1099

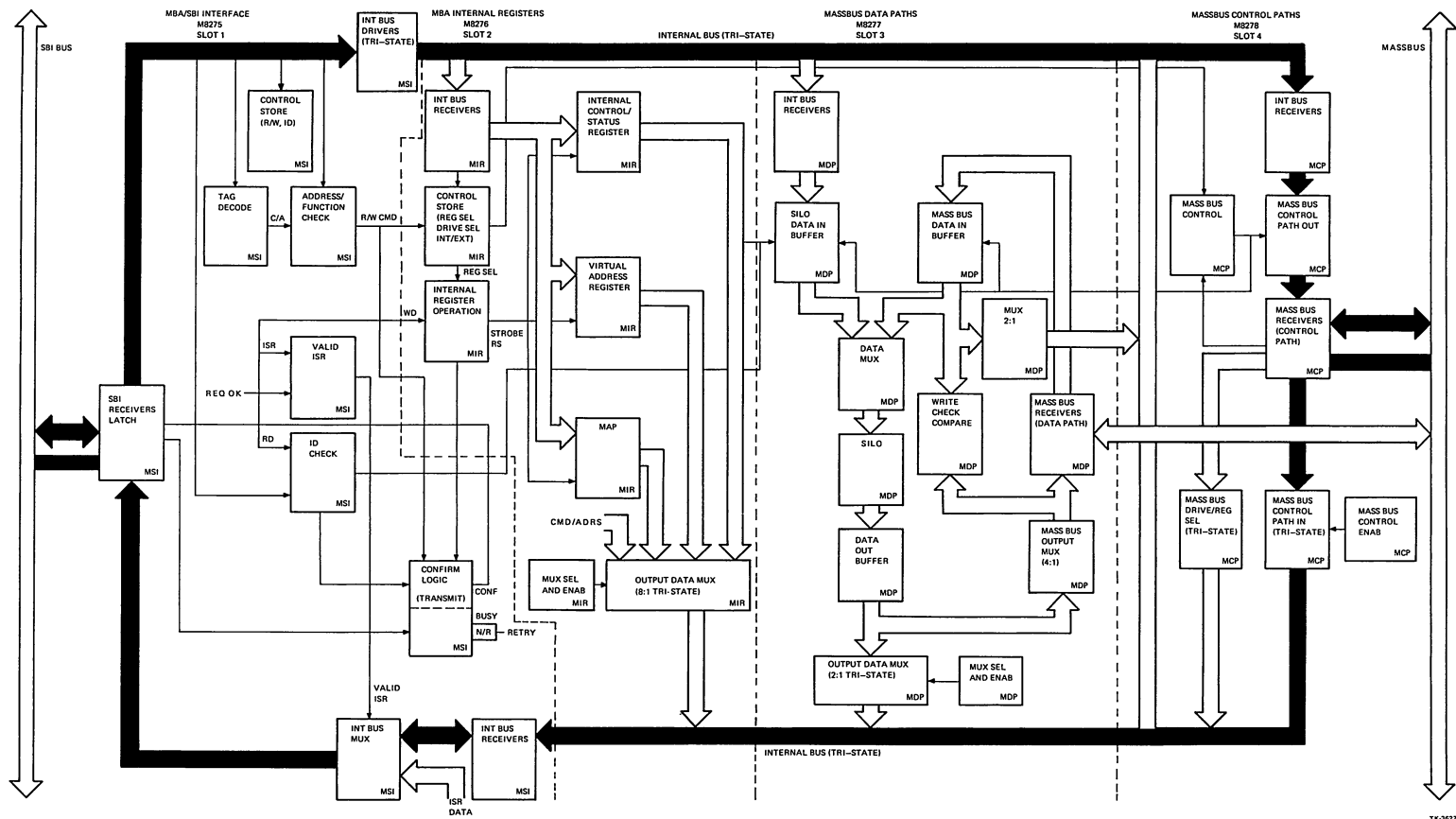
MBA Byte Counter Register



TK-1098

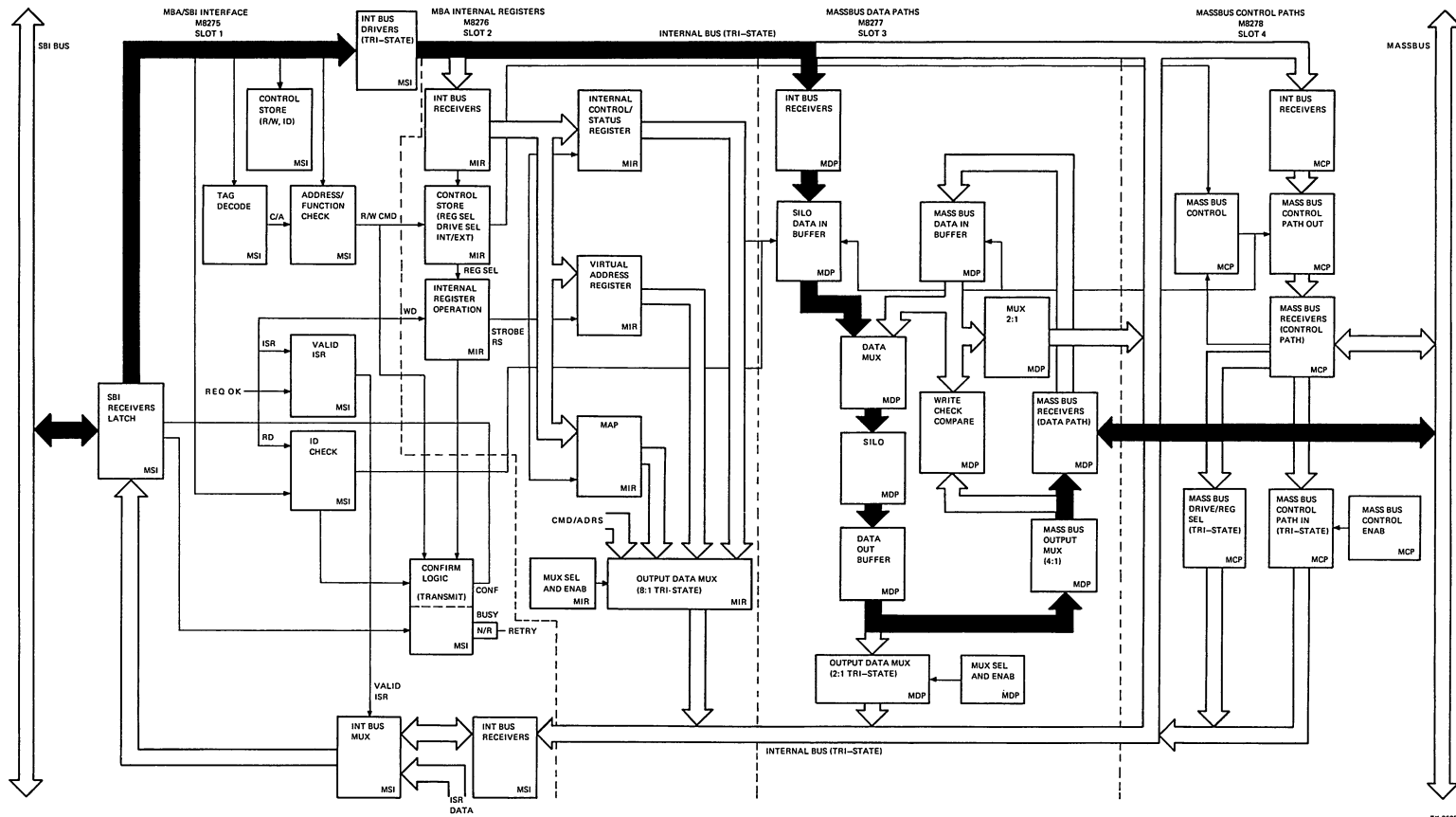
Diagnostic MASSBUS Control and Status Register





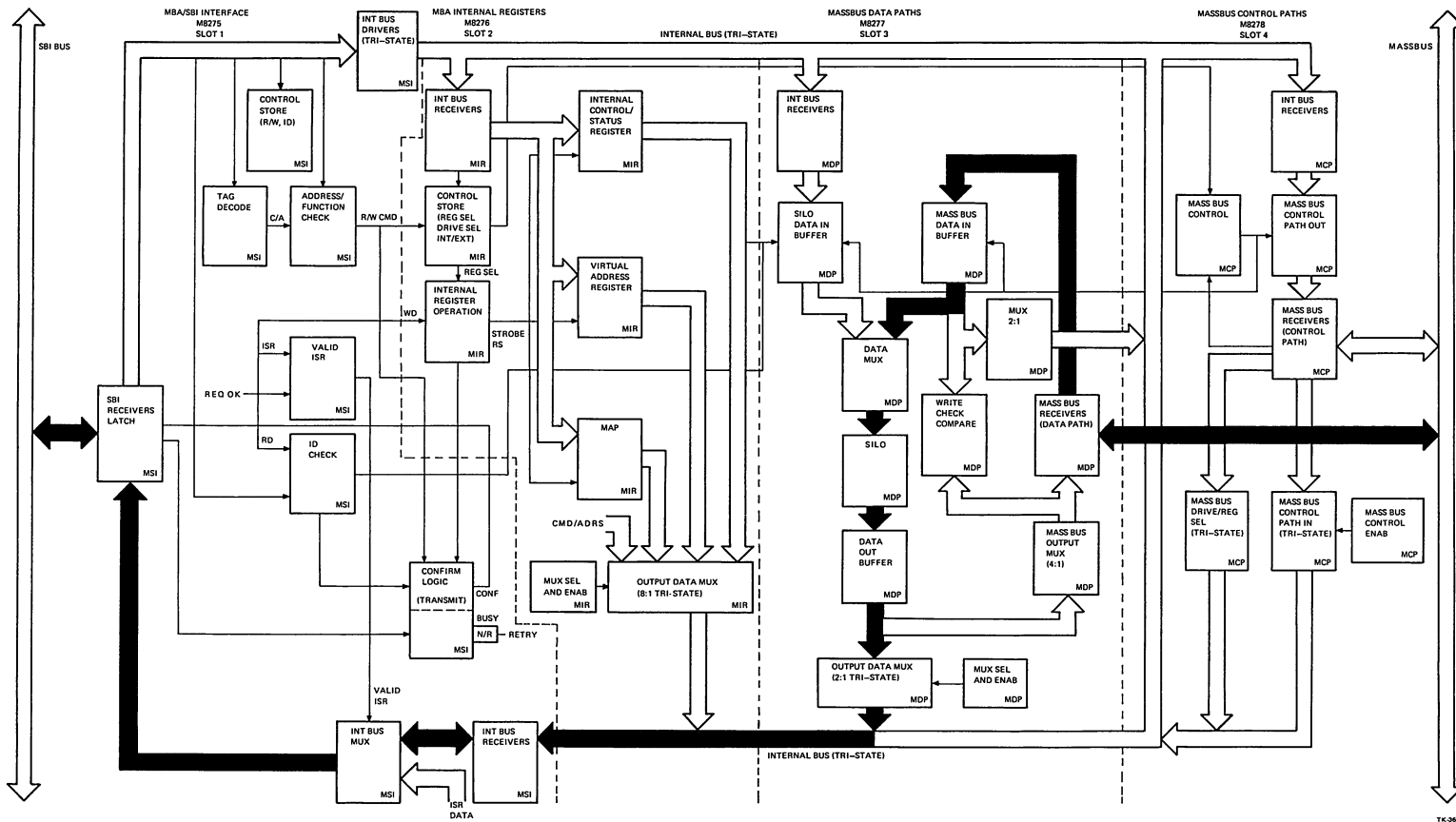
TK-3827

External Register Read/Write



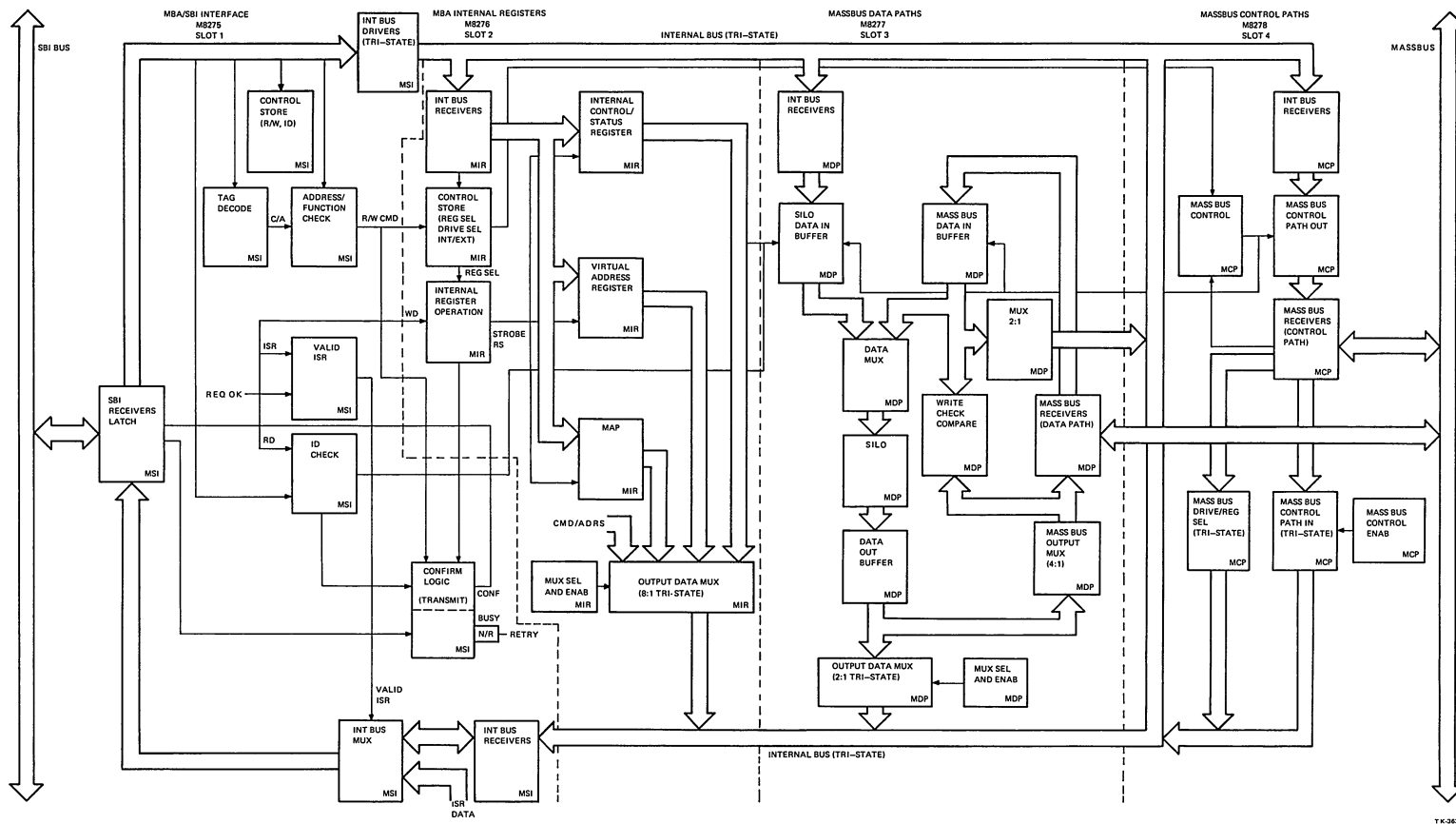
TK-3628

Data Transfer, Memory to Device



TK-3629

Data Transfer, Device to Memory



TK-3630

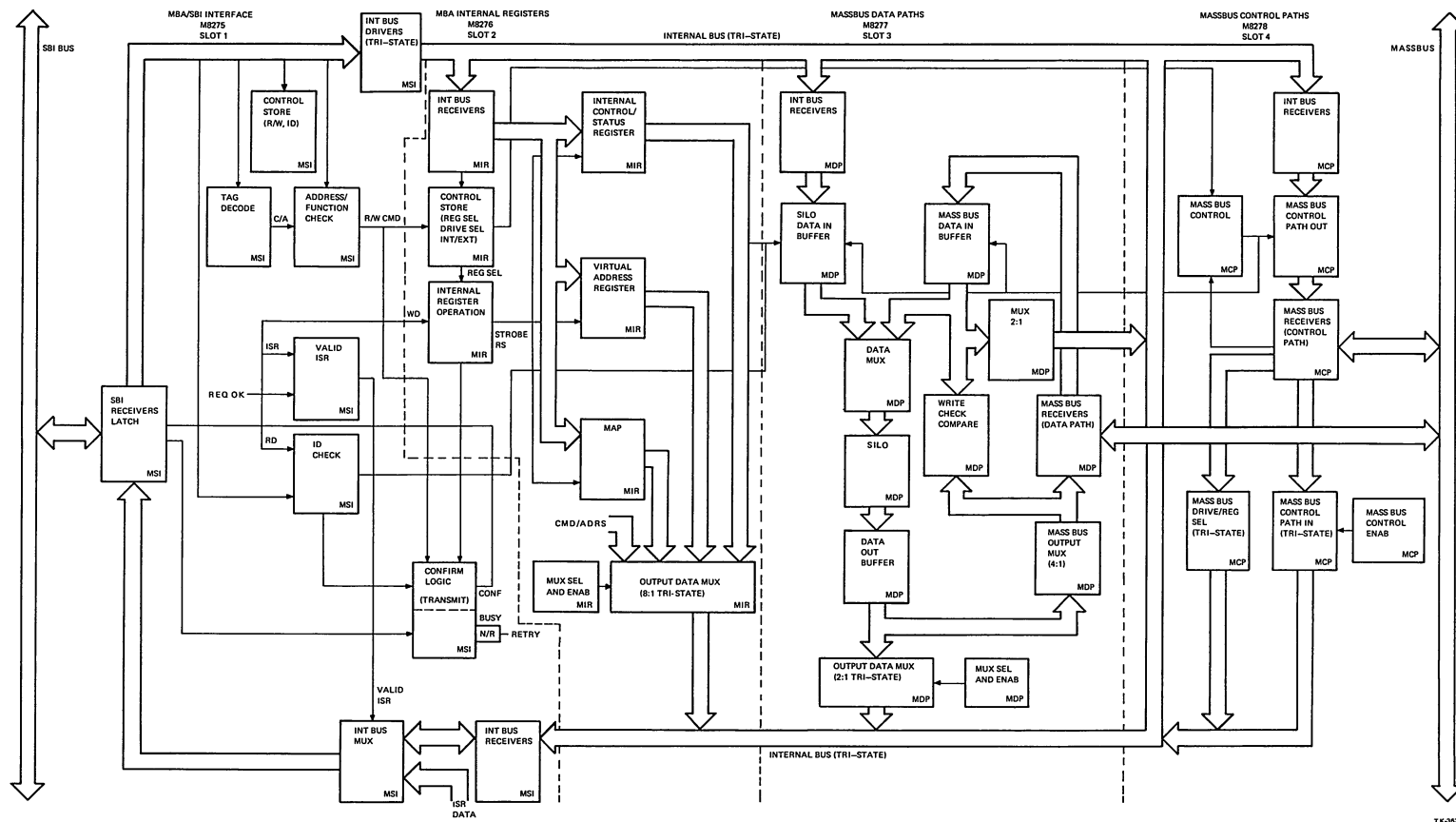
MBA Functional Block Diagram

BLANK

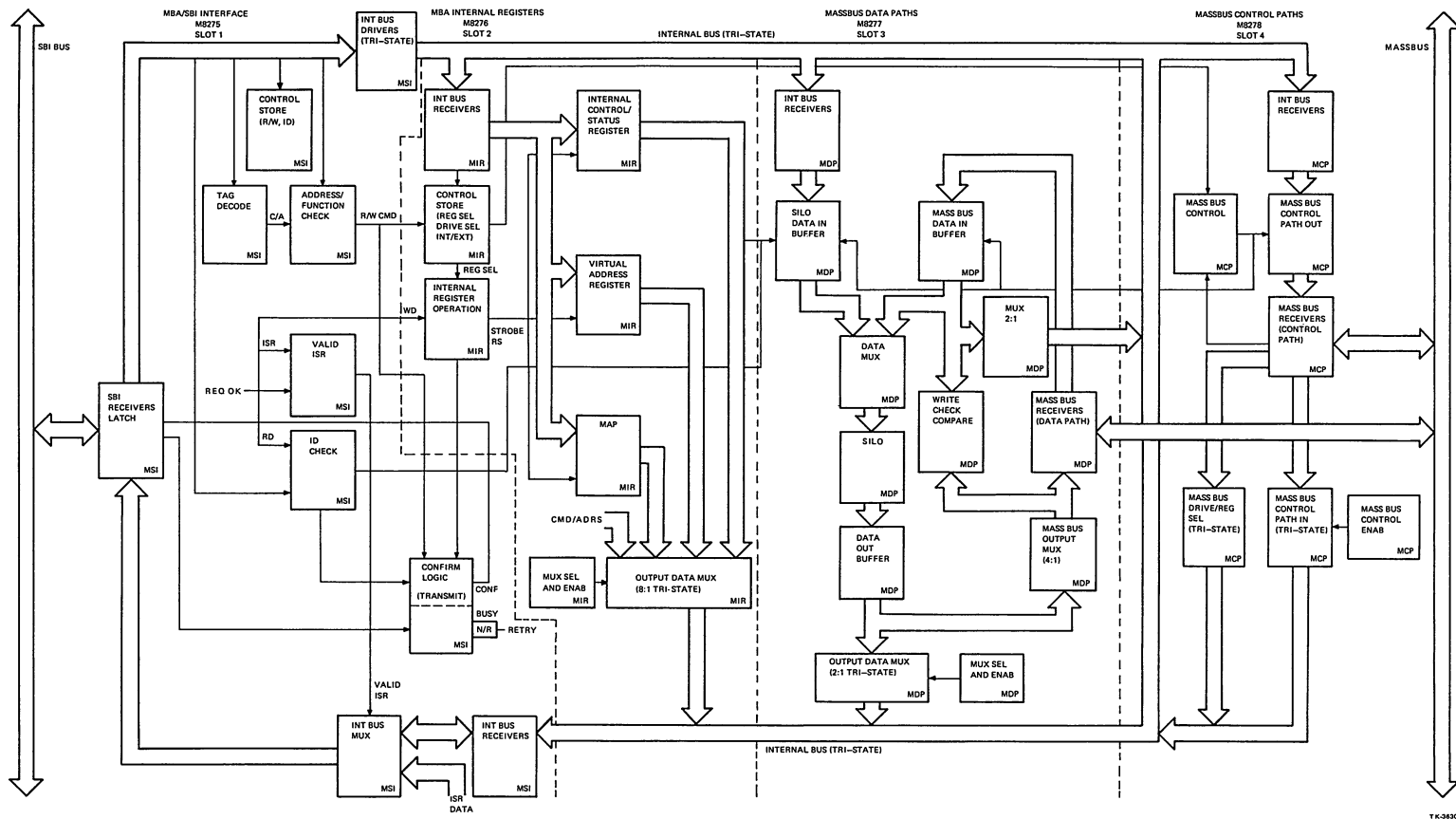
MODULE TEST

On the supplied copies of the MASSBUS functional block diagram, draw the data path needed to perform the following types of data transfer between the SBI and MASSBUS.

1. VAX-11/780 reads a register on a MASSBUS device.
2. VAX-11/780 reads the configuration register in the MBA.



MBA Functional Block Diagram



BLANK

UNIBUS ADAPTER

INTRODUCTION

This lesson will introduce you to the UNIBUS adapter, which serves as the interface for all UNIBUS devices. The lecture will acquaint you with all the features, while the lab projects will allow you to apply classroom knowledge to real situations.

OBJECTIVES

1. Given the following function:
 - a. Read/write buffered data path transfer
 - b. Read/write direct data path transfer
 - c. Internal register read/write
 - d. External register read/writetrace it on a functional block diagram of the UNIBUS adapter.
2. Identify UNIBUS adapter functions that require or do not require the use of the microsequencer.
3. Given a UNIBUS adapter function that requires the microsequencer, list the sequence of microstates necessary to perform it.

SAMPLE TEST ITEM

Select from the list below all those functions that require the aid of the microsequencer by placing an X to the left of the function:

- ☐ Read configuration register
- ☐ Read any BRRVR register
- ☐ Write any map register
- ☐ Write status register
- ☐ Read the control register
- ☐ Write configuration register
- ☐ Read any map register
- ☐ Read status register
- ☐ Write control register

RESOURCES

1. DW780 UNIBUS Adapter Technical Description

LECTURE OUTLINE

- I. Introduction
- II. Basic Hardware Description
- III. UNIBUS Adapter Overview
- IV. Reliability and Diagnostic Features
- V. UNIBUS Adapter Register Overview
- VI. Registers Read/Write Operations - Nonmicrosequencer
- VII. Microsequencer Overview
- VIII. Register Read/Write Operations - Microsequencer
- IX. Block Data Transfer (DMA)
- X. UNIBUS Device Interrupt Transaction Sequence
- XI. UBA Lab Projects

BLANK

EXERCISES

UBA Registers

Listed below are several versions of CNFGR contents and a list of possible error conditions. Match the CNFGR contents with the error condition.

- | CNFGR | | | Error Condition |
|-------|----------|-------------|-----------------------------|
| 1. | 10410024 | B pg. 2-111 | A. No errors |
| 2. | 0C410024 | D pg. 2-111 | B. Interlock sequence fault |
| 3. | 00410024 | A | C. UB power down |
| 4. | 00420024 | C pg. 2-112 | D. XMT fault, MXT fault |
5. If an environmental status bit in the CNFGR is set, what other bit must be set to cause an SBI interrupt request?
- The _____ bit in the _____ register. Pg. 2-115.
6. If the UNIBUS has 64K words of memory, the MRD field in the control register will be set to
- a. 00111₂
 - b. 01000₂
 - c. 10000₂
 - d. 10111₂
7. In question 6 above, which of the map registers will be disabled? (Answers are in decimal.)
- a. 0-15
 - b. 0-90
 - c. 0-255
 - d. 0-375

EXERCISES

8. List the bits in the status register that will cause an SBI interrupt when the USEFIE bit is set in the control register.

Pg. 2-117

- | | |
|----------|----------|
| a. _____ | d. _____ |
| b. _____ | e. _____ |
| c. _____ | f. _____ |
| | g. _____ |

9. What are the three ways to clear the SUEFIE bit in the control register?

- a. _____
- b. _____
- c. _____

LAB EXERCISES

In this exercise, you will observe and follow the microsequencer through "INITIALIZE UBA" using single-step mode.

```
>>>I
>>>U
>>>H
```

```
>>>D/ID      21      62  (IRD STATE OF CPU)
```

```
>>>D      200      00018FD0
```

```
>>>D+      049F0000
```

```
>>>D+      00200060
```

```
>>>SE      S0
```

```
>>>S      200
```

```
                /START 200
```

```
CPU CLOCK STOPPED
CPTO  UPC=0062
```

```
>>>S      S      B
```

```
>>>N
```

This exercise should reinforce your concept of the microsequencer during UBA INIT.

LAB EXERCISES

In this exercise, you will observe the microsequencer on UBA during wraparound mode using single-step mode.

```

>>>I      /INIT
>>>U      /UNJAM
>>>H      /HALT
>>>D/ID    21          62  /SETS BREAK POINT REGISTER
>>>D      200          00088FD0
>>>D+      009F8000
>>>D+      B0200068
>>>D+      9F87658F
>>>D+      20100000
>>>D+      00
>>>SE      S0
>>>S      200
>>>SE      S      B

```

```

CPU CLOCK STOPPED
CPT0 UPC=62

```

```

>>>N

```

This exercise should reinforce your concept of wraparound mode.

LAB EXERCISES

In this exercise, you will observe the operation of the UBA microsequencer. The following program, WRITE TO MAP REGISTER 0, is loaded.

```

>>>I          /INIT
>>>U          /UNJAM
>>>H          /HALT
>>>D/ID    21      62          DEPOSIT BREAKPOINT
>>>D    200      00088FD0    /200 MOVL# X800000008, @# 2006800
                                200 HLT
>>>D+    009F8000
>>>D+    0020068
>>>SE    S0          SETS SOMM
>>>S      200
                                CPU CLOCK STOPPED
                                CPT0 UPC=0062
>>>SE    S      B          SETS STEP BUS
>>>N          USE SPACE BAR
                                CPU CLOCK STOPPED

```

This exercise should reinforce the lecture on writing maps.

LAB EXERCISES

1. Examine and record all UBA backplane jumper configurations.

Jumper	In	Out
W1 _____		
W2 _____		
W3 _____		
W4 _____		
W5 _____		
W6 _____		
W7 _____		
W8 _____		

2. Initialize the system.

3. Examine and record the following UBA registers.

a. CNFGR	
b. UACR	
c. UASR	
d. DCR	
e. FMER	
f. FUBAR	
g. BRSVR	
h. BRRVR	
i. DPR 0	
j. DPR 1	
k. MR 0	
l. MR 1	

LAB EXERCISES

4.
 - a. Load known data into an SBI memory location.
 - b. Map that location to UNIBUS memory space.
 - c. Examine that UNIBUS memory location for the known data.

Example: Load into SBI memory location 0 12345678. Set MR 0 to Valid, DDP, PFN 0. Examine UNIBUS address 0; it should be 5678.

SBI memory address _____ Contents _____

Map register contents _____

UNIBUS address contents _____ (should be equal to SBI address contents)

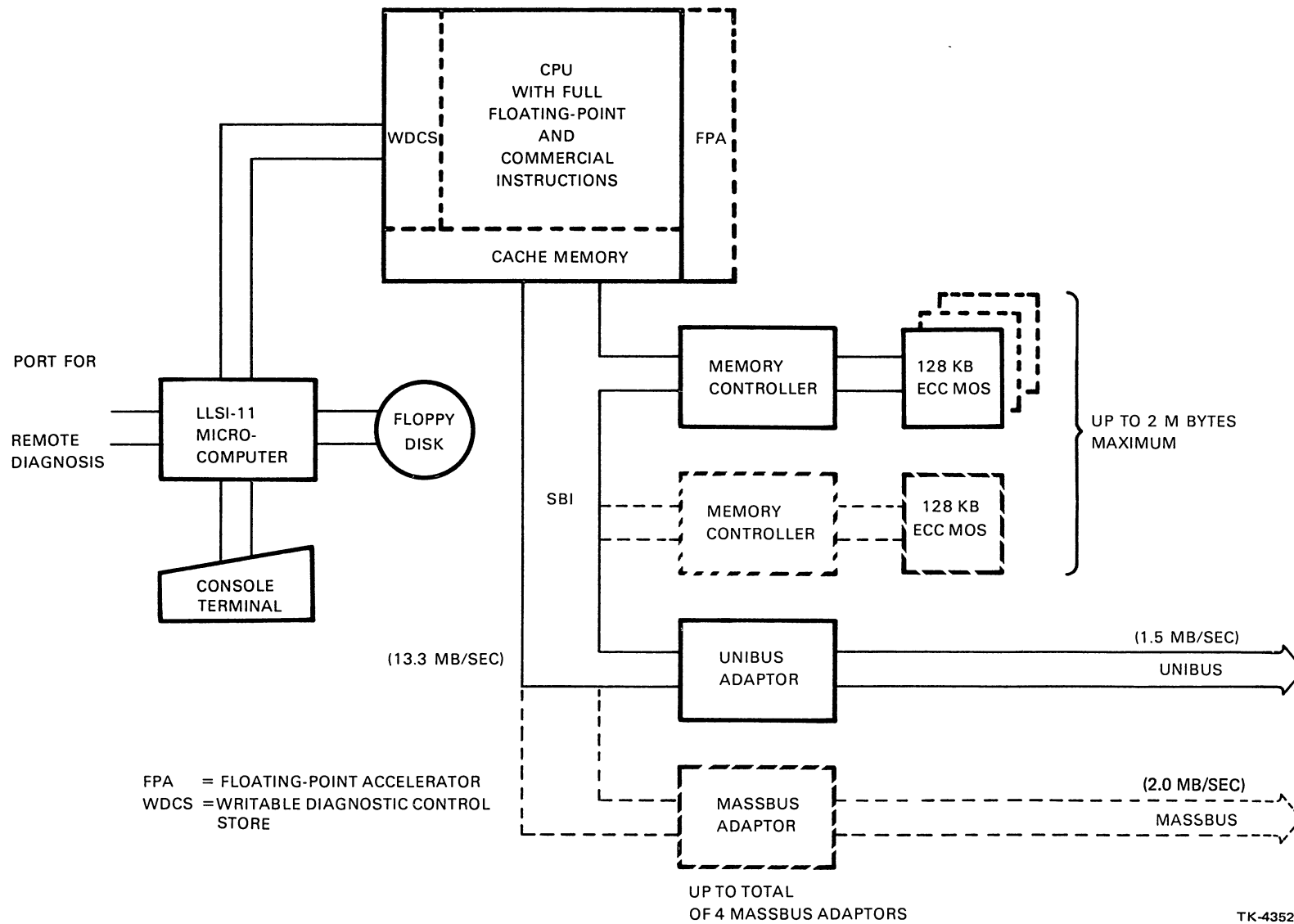
5. Do item 4 above for both DDP and one of the BDPs.

SBI memory address _____ Contents _____

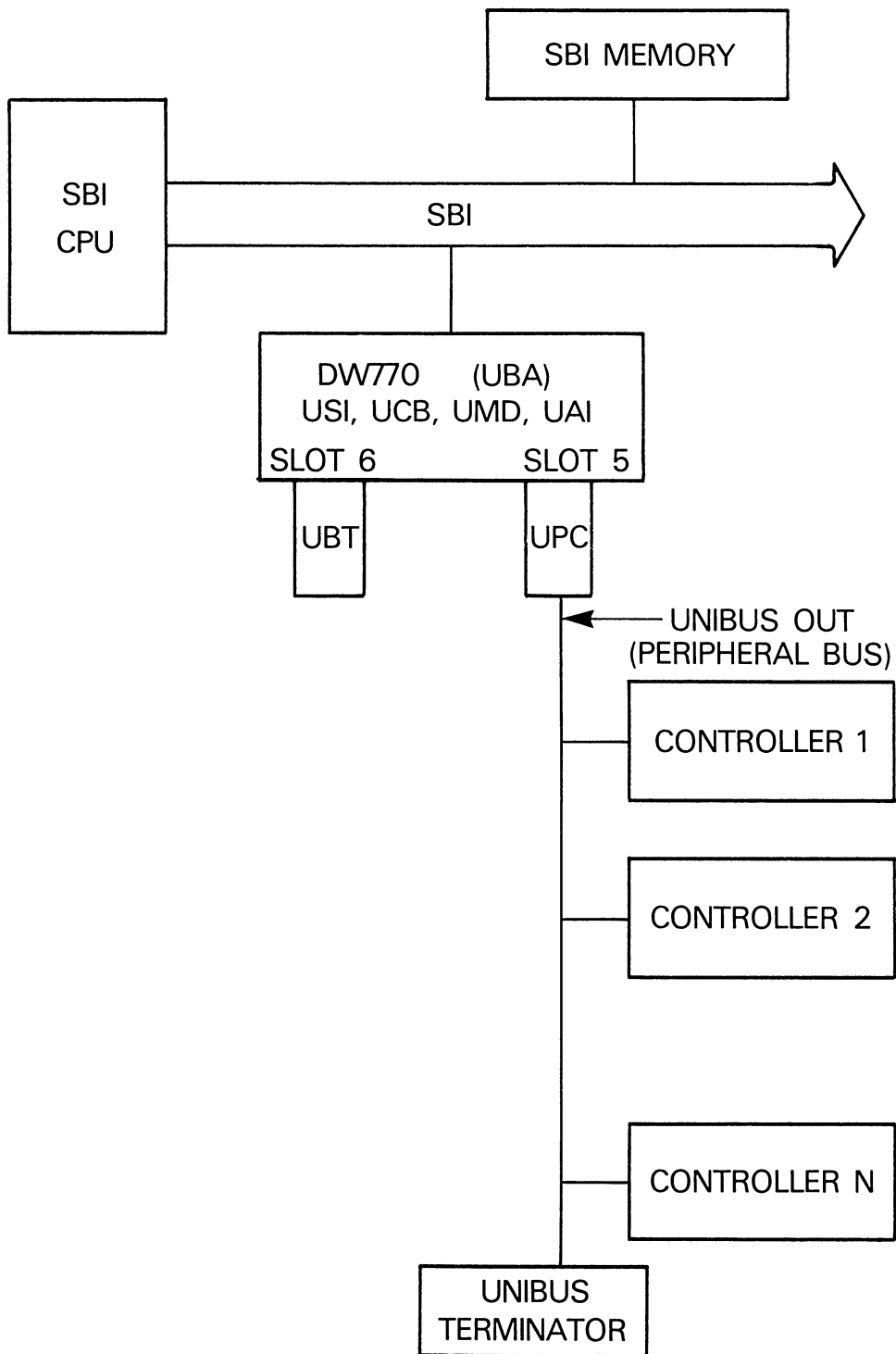
Map register contents _____

UNIBUS address contents _____

BLANK

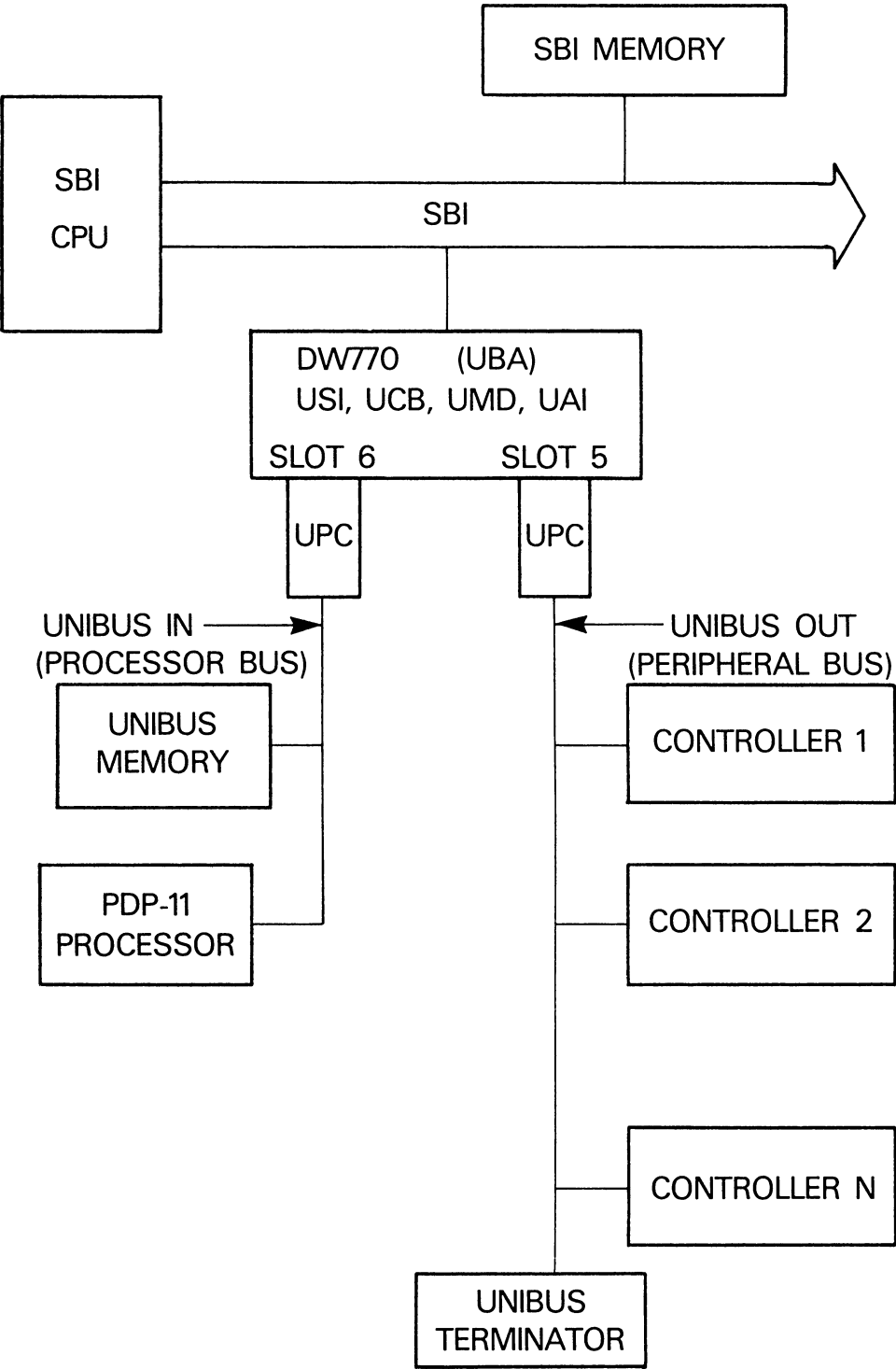


11/780 System



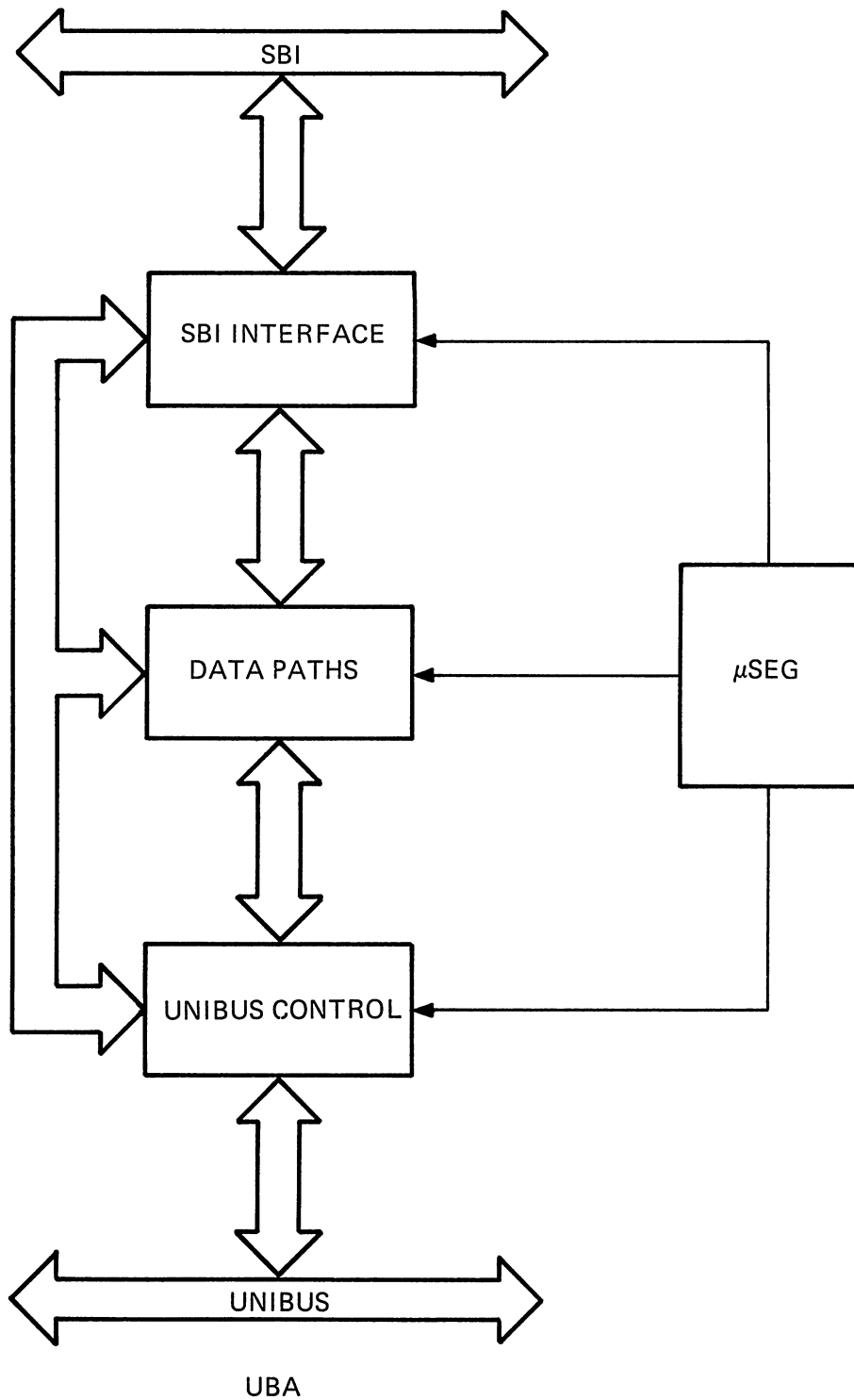
TK-3606

UNIBUS Subsystem Configuration



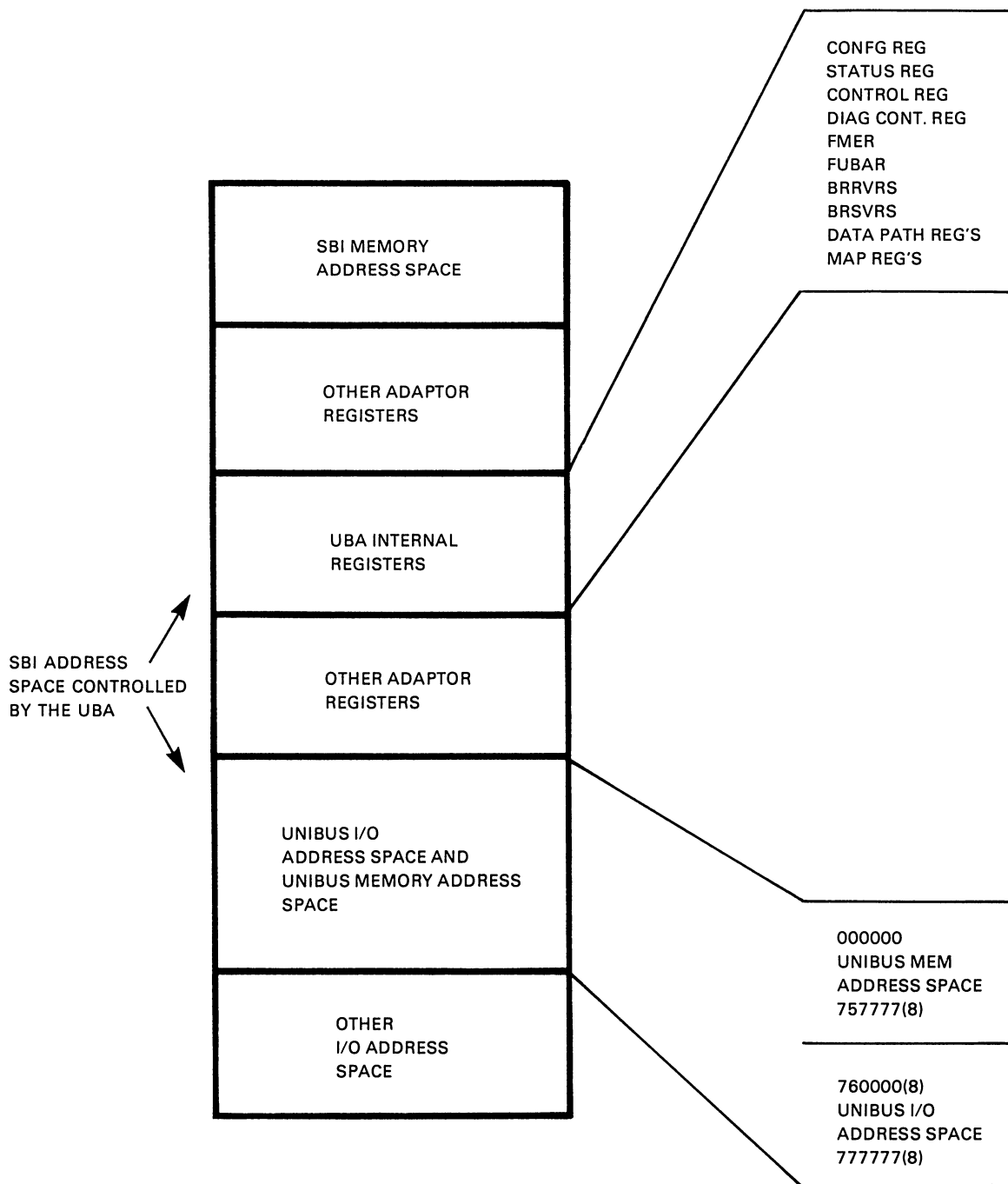
TK-3607

UBA and PDP-11 Subsystem Configuration



TK-3604

UBA Simplified Block Diagram



TK-3602

SBI Address Space

UBA #	DEVICE	UNIBUS ADDRESS (OCTAL)	SBI ADDRESS (HEX)	11/780 PHYSICAL ADDRESS (HEX)	VECTORS (HEX)
0	CR11/CRS1	777160	0804FF9C	2013FE70	98
0	CR11/CRB1	777162	0804FF9C	2013FE72	
0	CR11/CRB2	777164	0804FF9D	2013FE74	
0	LP11/LPST	777514	0804FFD3	2013FF4C	80
0	LP11/LPDR	777516	0804FFD3	2013FF4E	
0	RK611 CS1	777440	0804FFC8	2013FF20	
0	RK611 WC	777442	0804FFC8	2013FF22	(TBS)
0	RK611 BA	777444	0804FFC9	2013FF24	
0	RK611 DA	777446	0804FFC9	2013FF26	
0	RK611 CS2	777450	0804FFCA	2013FF28	
0	RK611 DS	777452	0804FFCA	2013FF2A	
0	RK611 ERR	777454	0804FFCB	2013FF2C	
0	RK611 A&O	777456	0804FFCB	2013FF2E	
0	RK611 CYL	777460	0804FFCC	2013FF30	
0	UNUSED	777462	0804FFCC	2013FF32	
0	RK611 DB	777464	0804FFCD	2013FF34	
0	RK611 MR1	777466	0804FFCD	2013FF36	
0	RKECPS	777470	0804FFCE	2013FF38	
0	RKECPT	777472	0804FFCE	2013FF3A	
0	RK611 MR2	777474	0804FFCF	2013FF3C	
0	RK611 MR3	777476	0804FFCF	2013FF3E	
0	DZ11	FLOATING ADDRESSES AND VECTORS			
0	DMC11	ACCORDING TO PDP-11			
0	DR11-B	CONVENTION			

TK-3850

Addresses and Vectors for UNIBUS Devices

PDP-11 = 760000₈-777777₈

VAX-11 = 2013E000₁₆-2013FFFF₁₆

EXAMPLE

			PDP-11	VAX-11
LP11	LPS =	777514 ₈	OR	2013FF4C ₁₆
LP11	LPB =	777516 ₈	OR	2013FF4E ₁₆

SIMPLIFIED CONVERSION

7	7	7	5	1	4	
1 1 1	1 1 1	1 1 1	1 0 1	0 0 1	1 0 0	OCTAL
3	F	F	4		C	BINARY
						HEX

CONVERT THE OCTAL REGISTER ADDRESS TO HEX AND ADD 20100000₁₆

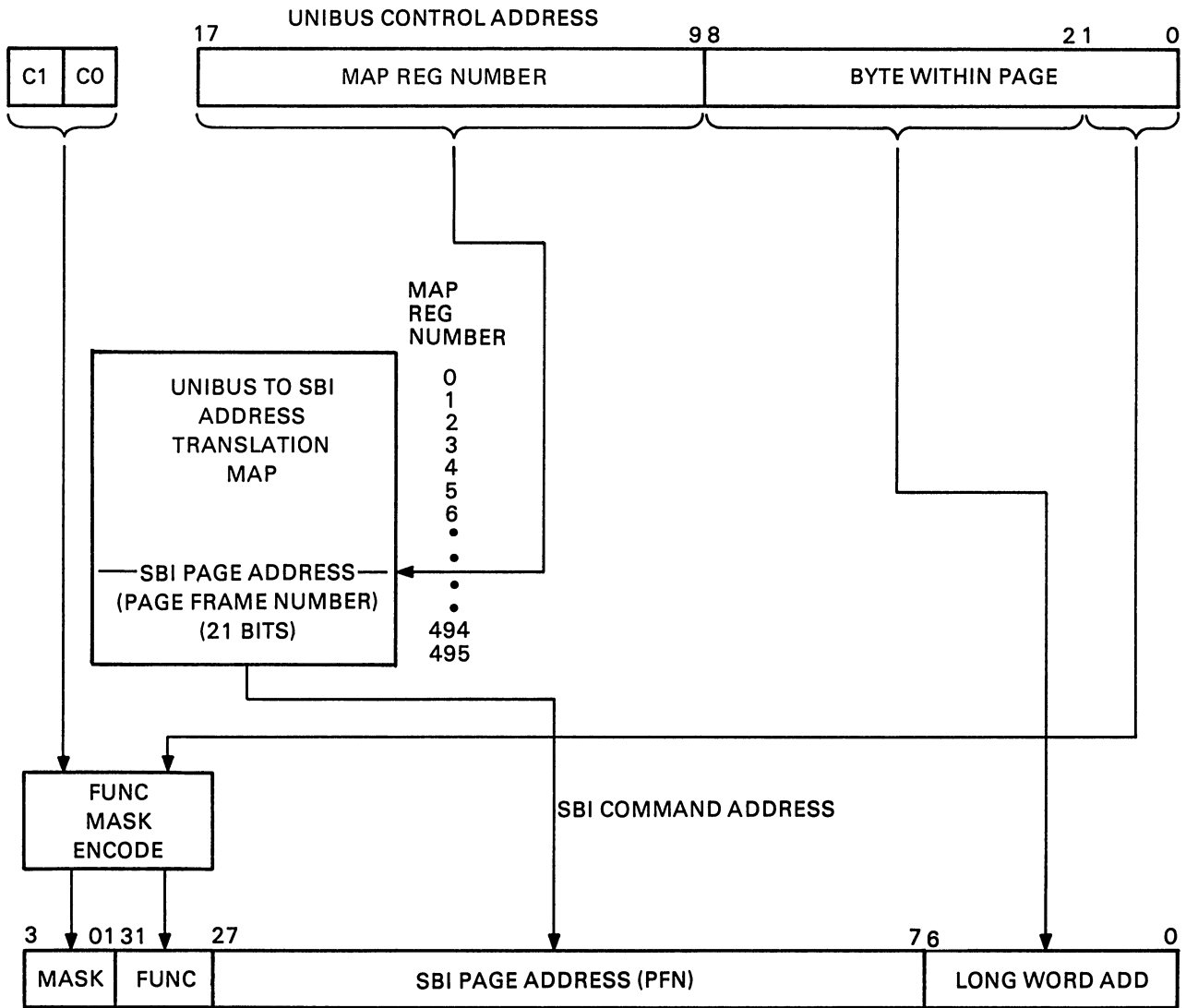
TK-3852

UNIBUS Device Register Address Translation

SBI		UNIBUS	
FUNCTION	MASK	CONTROL	ADDRESS
3:0	3 2 1 0	C 1:0	1:0
READ MASKED	0 0 0 1	DATI	0 0
	0 0 1 1	DATI	0 0
	0 0 1 0	DATI	0 0
	0 1 0 0	DATI	1 0
	1 1 0 0	DATI	1 0
	1 0 0 0	DATI	1 0
WRITE MASKED	0 0 0 1	DATOB	0 0
	0 0 1 0	DATOB	0 1
	0 1 0 0	DATOB	1 0
	1 0 0 0	DATOB	1 1
	0 0 1 1	DATO	0 0
	1 1 0 0	DATO	1 0
INTERLOCK READ MASKED (SETS INTERLOCK FLIP-FLOP FOR DATIP-DATO SEQUENCE)	0 0 0 1	DATIP	0 0
	0 0 1 0	DATIP	0 0
	0 1 0 0	DATIP	1 0
	1 0 0 0	DATIP	1 0
	0 0 1 1	DATIP	0 0
	1 1 0 0	DATIP	1 0
INTERLOCK WRITE MASKED	0 0 0 1	DATOB	0 0
	0 0 1 0	DATOB	0 1
	0 1 0 0	DATOB	1 0
	1 0 0 0	DATOB	1 1
	0 0 1 1	DATO	0 0
	1 1 0 0	DATO	1 0

TK-3577

SBI Function-Mask Translation to UNIBUS Control-Address



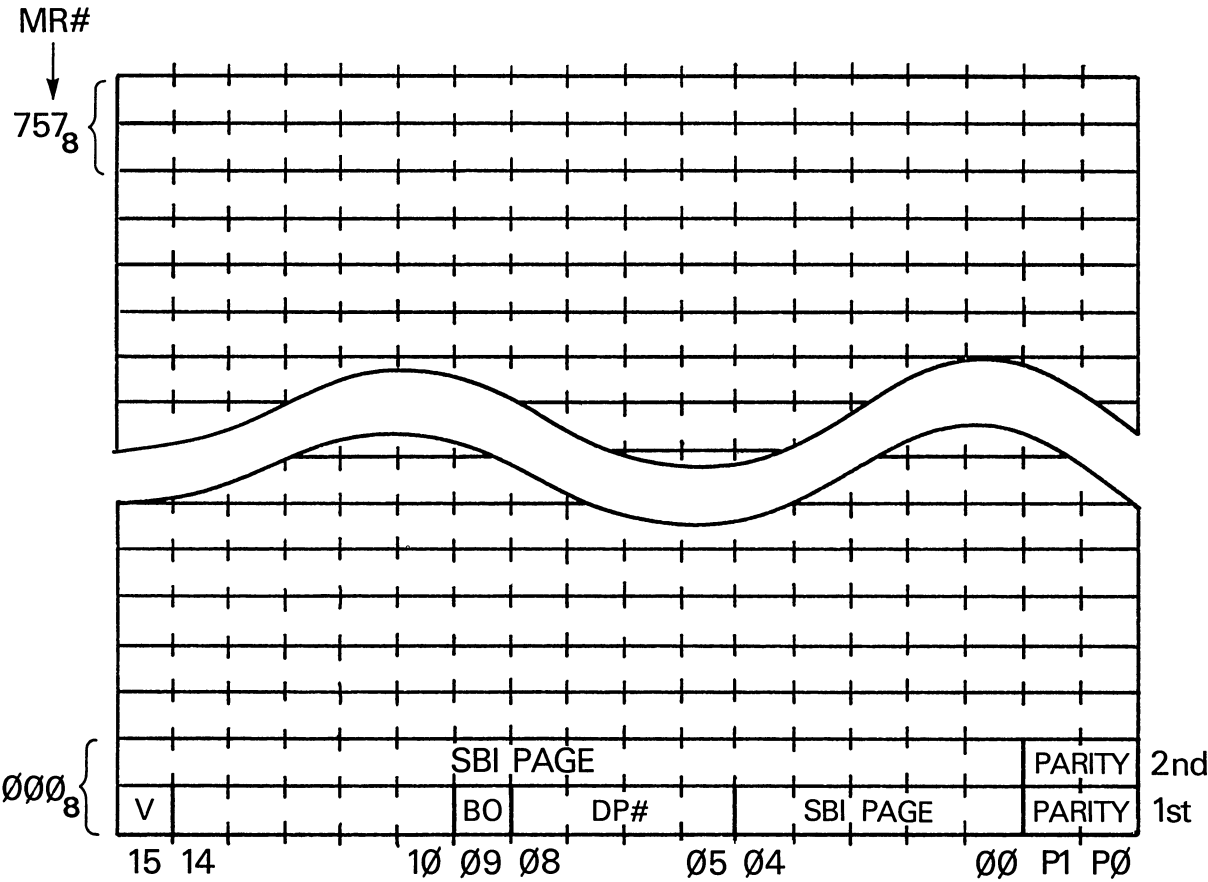
TK-0151

UNIBUS to SBI Address Translation

UNIBUS		SBI	
C 1:0	A 1:0	FUNC 3:0	MASK 3 2 1 0
DATI	0 0	READ MASKED	0 0 1 1
	1 0		1 1 0 0
DATO	0 0	WRITE MASKED	0 0 1 1
	1 0		1 1 0 0
DATOB	0 0	WRITE MASKED	0 0 0 1
	0 1		0 0 1 0
	1 0		0 1 0 0
	1 1		1 0 0 0
DATIP	0 0	INTERLOCK READ MASKED	0 0 1 1
	1 0		1 1 0 0
FOLLOWED BY DATO	0 0	INTERLOCK WRITE MASKED	0 0 1 1
	1 0		1 1 0 0
OR DATOB	0 0	INTERLOCK WRITE MASKED	0 0 0 1
	0 1		0 0 1 0
	1 0		0 1 0 0
	1 1		1 0 0 0

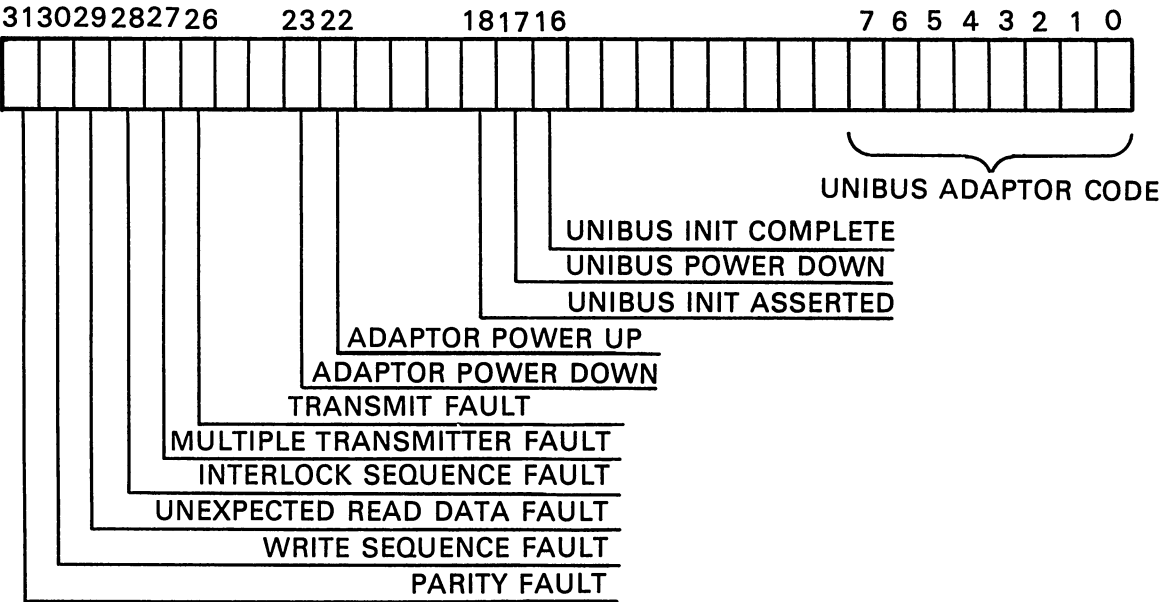
TK-3590

UNIBUS Control-Address to SBI Function-Mask Encoder



TK-3605

Map (1K x 18 RAM)



TK-0119

UBA Configuration Register
(UBACNFR)
Offset = 000

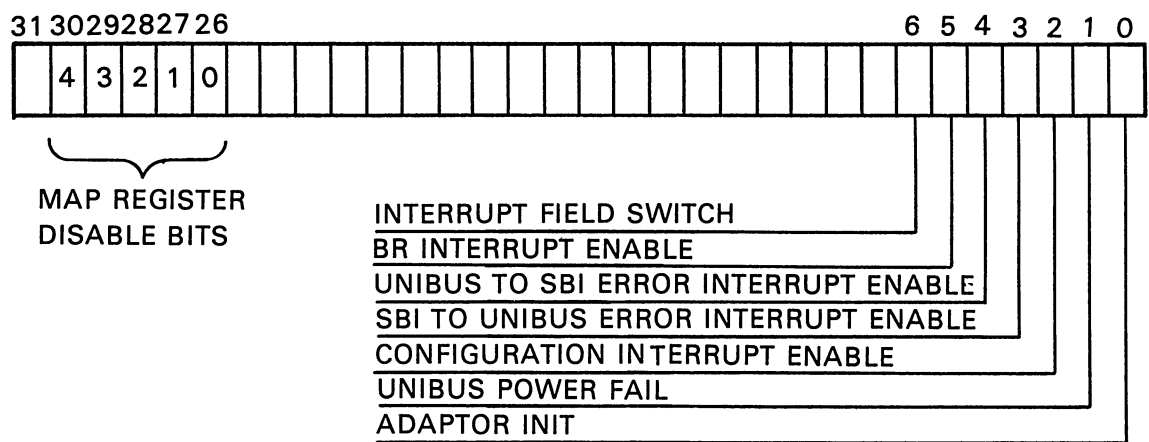
UBA ADAPTOR CODE									
BITS	7	6	5	4	3	2	1	0	
	0	0	1	0	1	0	V	V	
							B	A	
UBA NUMBER		0				0		0	ADAPTOR CODE 28 ₁₆
		1				0		1	29
		2				1		0	2A
		3				1		1	2B

The adaptor code is determined by backplane jumpers and indicates the starting address of the UNIBUS address space associated with each UBA.

UBA NUMBER (VB, VA)	STARTING ADDRESS OF THE UNIBUS ADDRESS SPACE (PHYSICAL BYTE ADDRESS)	ADAPTOR CODE (16)
0	2 0 1 0 0 0 0 0	28
1	2 0 1 4 0 0 0 0	29
2	2 0 1 8 0 0 0 0	2A
3	2 0 1 C 0 0 0 0	2B

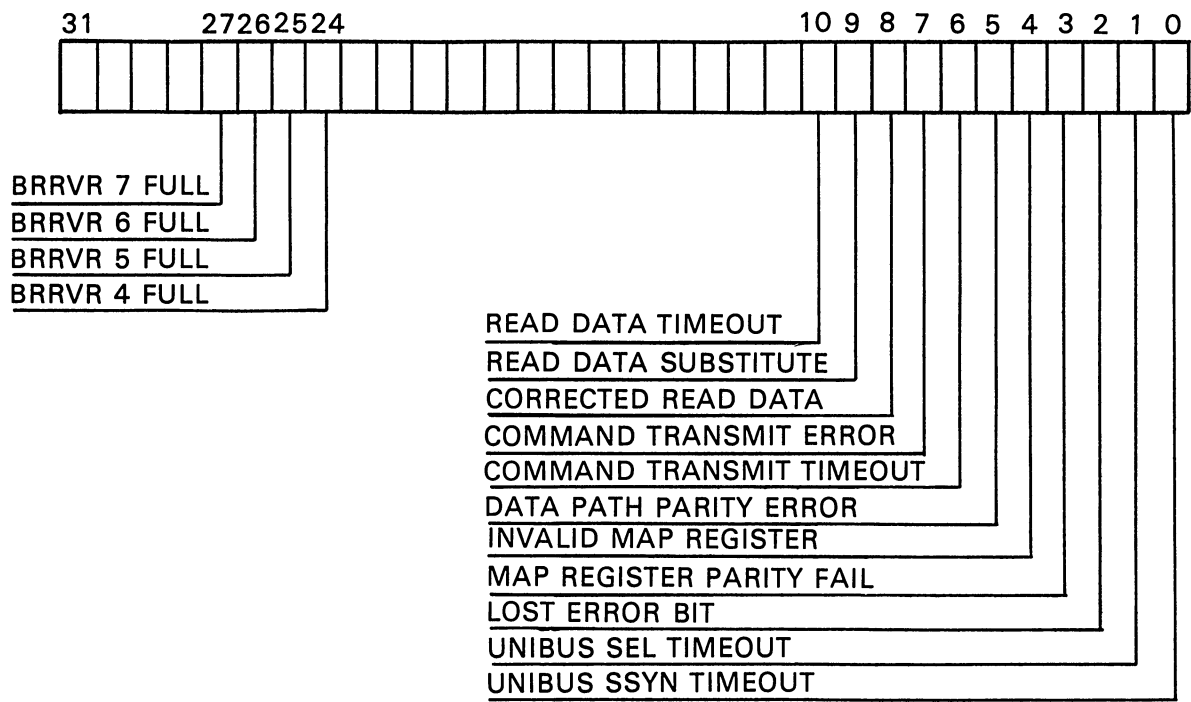
TK-3596

CNFGR Register Bits 7:0



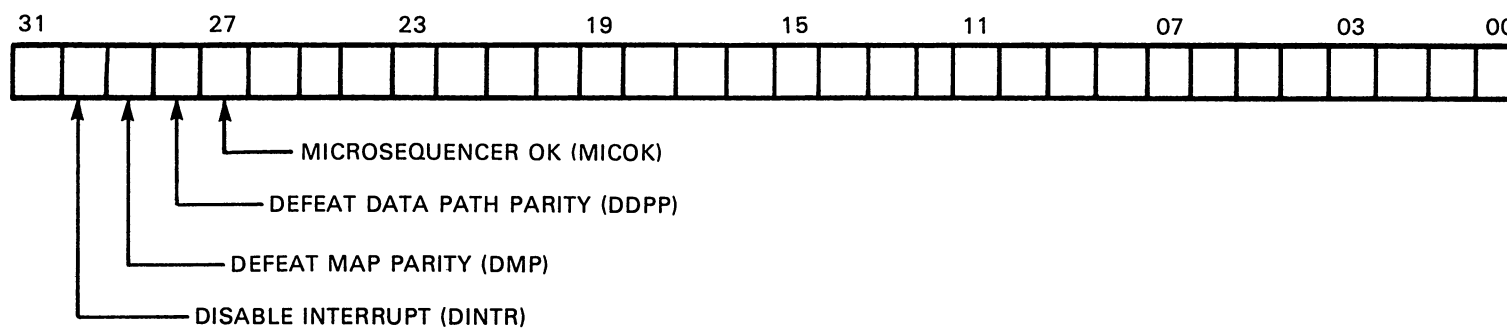
TK-0120

Map Register Disable Bits



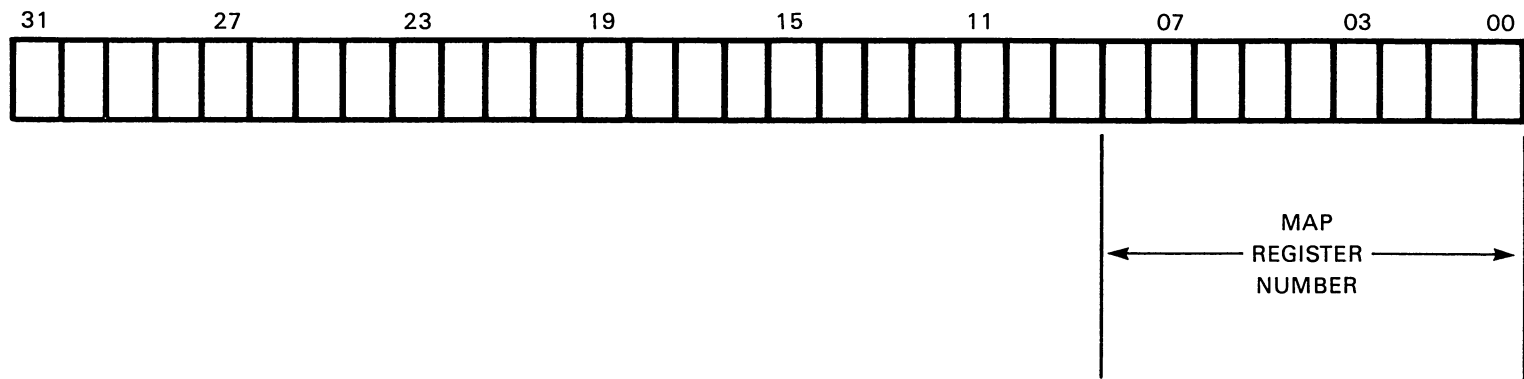
TK-0121

UBA Status Register
(UBASR)
Offset = 008



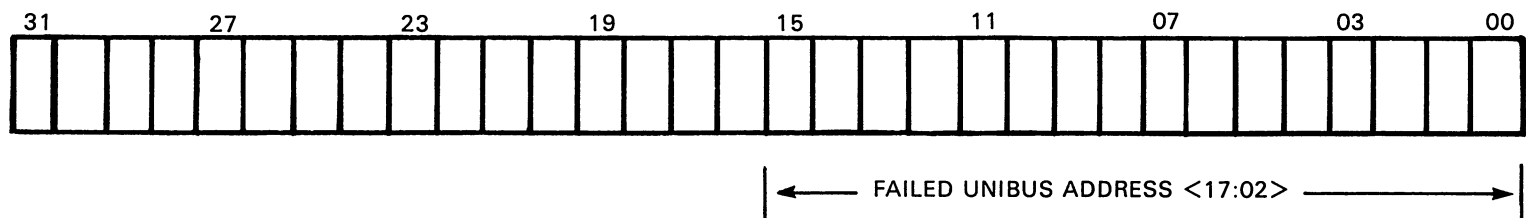
TK-3579

UBA Diagnostic Control Register
(DCR)
Offset = 00C



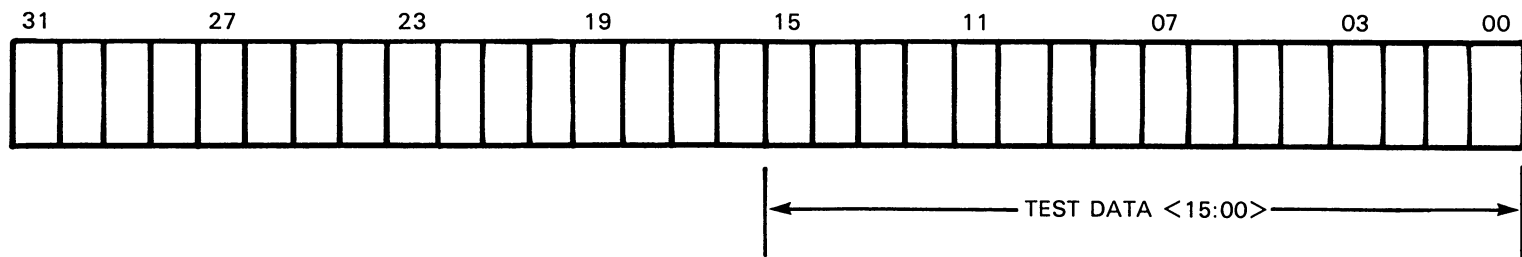
TK-3578

Failed Map Entry Register
(FMER)
Offset = 010



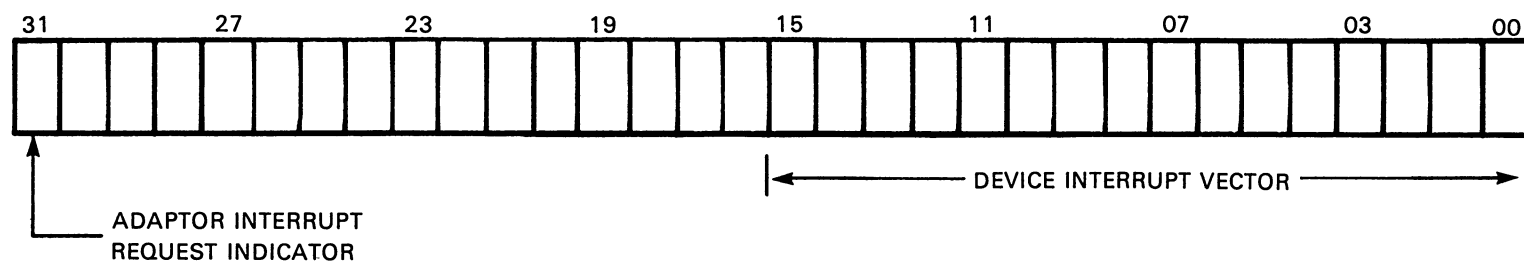
TK-3583

Failed UNIBUS Address Register
(FUBAR)
Offset = 014



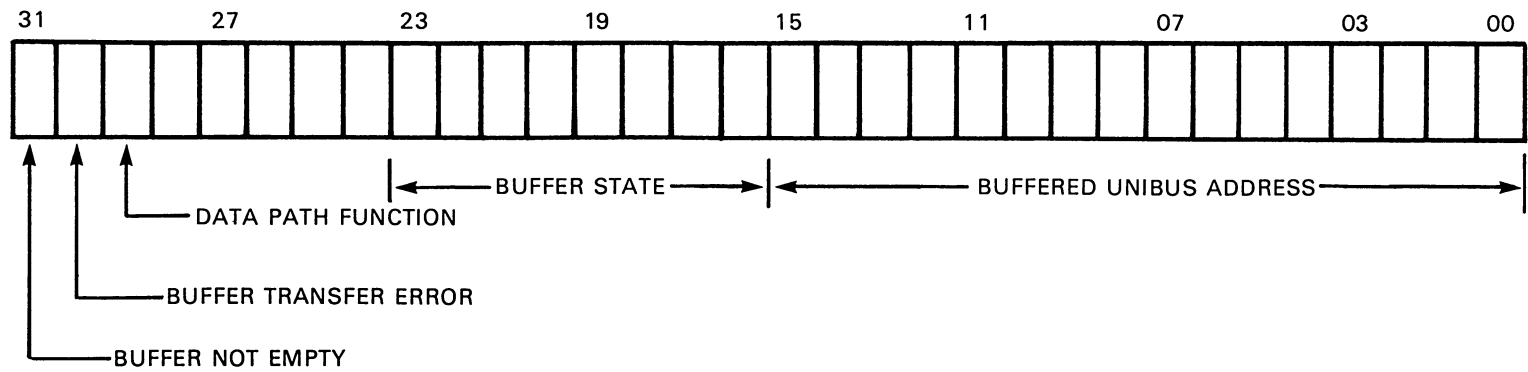
TK-3581

Buffer Selection Verification Register 0 through 3
 (BRSVR 0,1,2,3)
 Offset = 020,024,028,02C



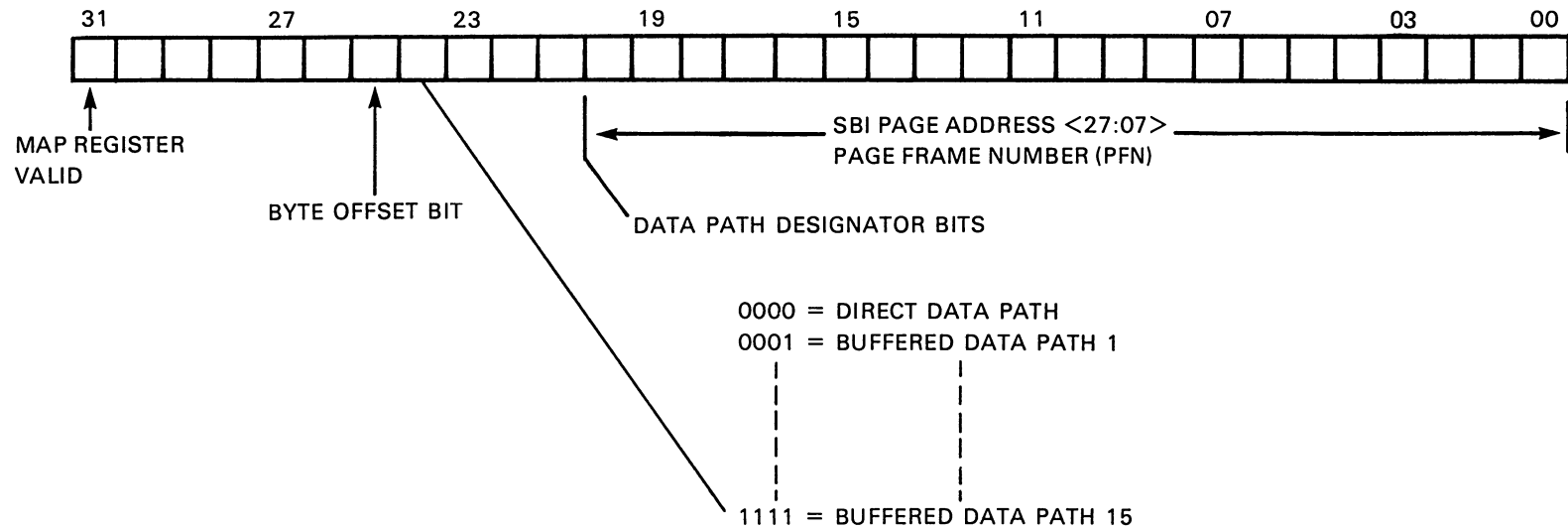
TK-3580

BR Receive Vector Registers 4 through 7
(BRRVR 4,5,6,7)
Offset = 030,034,038,03C



TK-3585

Data Path Registers 0 through 15
(DPR 0 through 15)
Offset = 040 through 07C



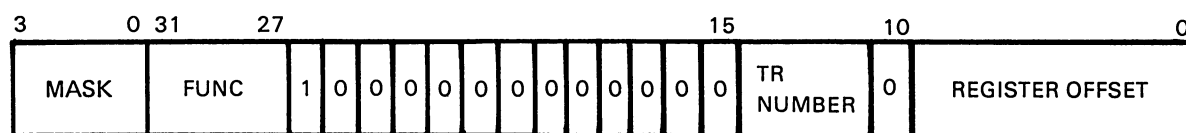
TK-3582

Map Registers
 (MR 0 through 495)
 Offset = 800 through FBC

REGISTER	FUNCTION	NEED OF MICRO- SEQUENCER	COMMENTS
CNFG	READ WRITE	NO NO	
UBACR	READ WRITE	YES NO	
UBASR	READ WRITE	YES NO	
DCR	READ WRITE	NO NO	
FMER	READ/ONLY	YES	SBI ERROR CNF ON WRITE
FUBAR	READ/ONLY WRITE		SBI ERROR CNF ON WRITE
BRRVR S	READ/ONLY	YES	READ ONLY REGISTERS
BRSVR'S	READ WRITE	YES YES	
DRR S	READ WRITE	YES YES	BIT 31 ONLY
MAP REG	READ WRITE	YES YES	

TK-4351

Microsequencer Functions
for Register



BASE ADDRESSES

TRLBASE ADDRESS NUM	(PHYSICAL)
0	20000000
1	20002000
2	20004000
3	20006000
4	20008000
5	2000A000
6	2000C000
7	2000E000
8	20010000
9	20012000
10	20014000
11	20016000
12	20018000
13	2001A000
14	2001C000
15	2001E000

REGISTER OFFSETS

UBA REG	BYTE (PHYSICAL)
CNFG	000
UBACR	004
UBASR	008
DCR	00C
FMER	010
FUBAR	014
FMER	018
FUBAR	01C
BRSVR 0	020
BRSVR 1	024
BRSVR 2	028
BRSVR 3	02C
BRRVR 4	030
BRRVR 5	034
BRRVR 6	038
BRRVR 7	03C
DPR 0	040
DPR 1	044
DPR 14	078
DPR 15	07C
RESERVED	100
RESERVED	7FC

UBA		<u>OFFSET</u>
<u>REG</u>		
MR	0	800
MR	1	804
MR	494	FB8
MR	495	FBC
RESERVED		FC0
RESERVED		FFC

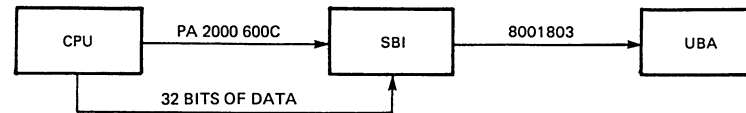
SBI Command Address Format for UBA Register Access

	UBA #0	
PA		SBI
2000 6000	CNF	8001800
004	CR	8001801
008	SR	8001802
00C	DCR	8001803
010	FMER	8001804
014	FUBAR	8001805
018	FMER	8001806
01C	FUBAR	8001807
020	BRSVR0	8001808
024	BBSVR1	8001809
028	BRSVR2	800180A
02C	BRSVR3	800180B
030	BRRVR4	800180C
034	BRRVR5	800180D
038	BRRVR6	800180E
03C	BRRVR7	800180F
040	DPR0	8001810
044	DPR1	8001811
048	DPR2	8001812
04C	DPR3	8001813
050	DPR4	8001814
054	DPR5	8001815
058	DPR6	8001816
05C	DPR7	8001817
060	DPR8	8001818
064	DPR9	8001819
068	DPR10	800181A
06C	DPR11	800181B
070	DPR12	800181C
074	DPR13	800181D
078	DPR14	800181E
07C	DPR15	800181F
800	MR0	8001A0
804	MR1	8001A1
	.	.
	.	.
	.	.
	.	.
	.	.
FB8	MR494	.
FBC	MR495	.

TK-3849

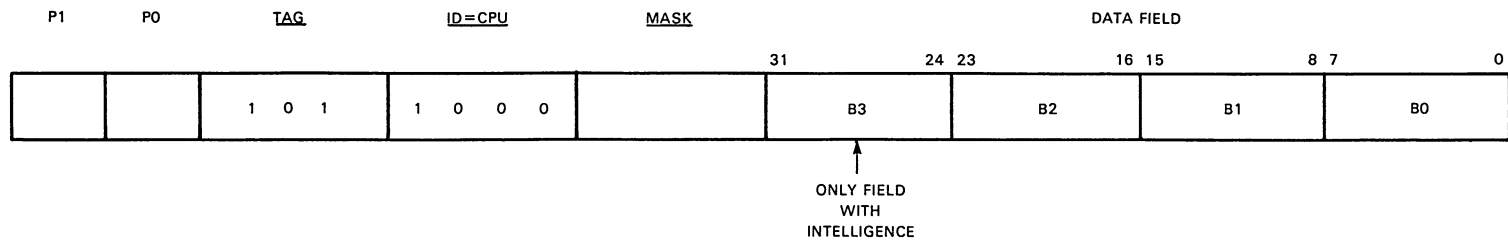
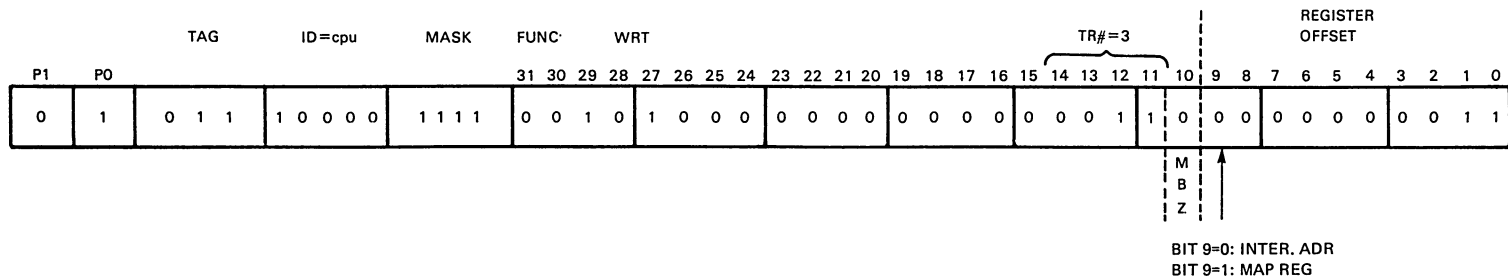
UBA #0 Register Addresses

WRITE TO DCR
DOES NOT ENVOKE
MICRO SEQUENCER



A	TR	TR0			
I		C/A	WD		
C				ACK	ACK

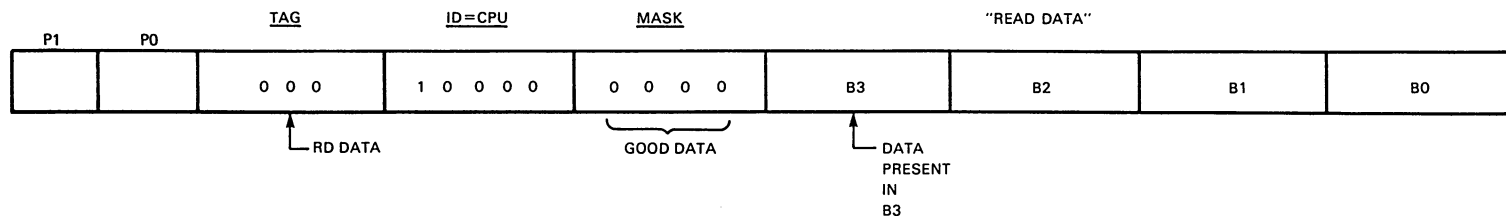
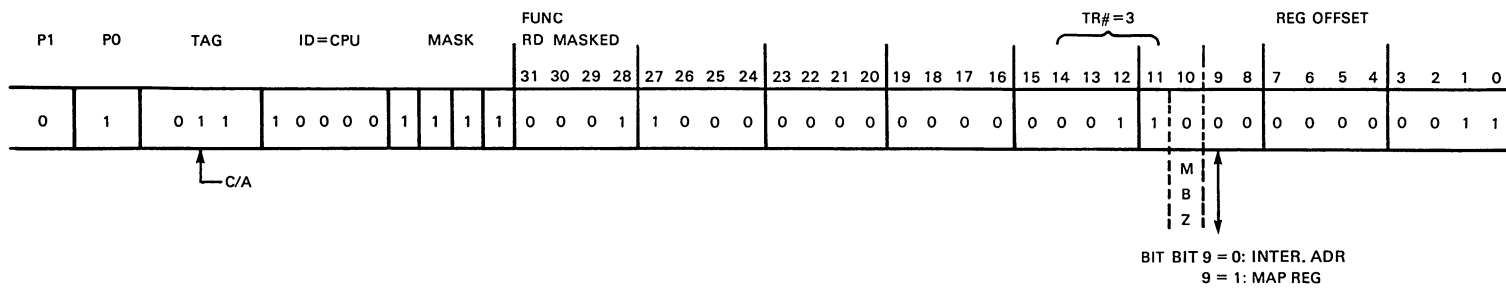
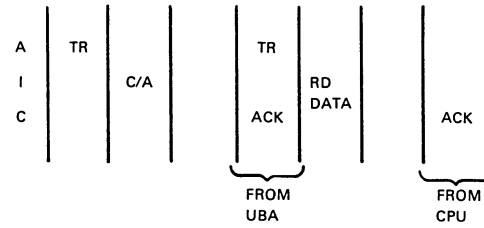
PA <29:02> = SBI <27:00>



Write to DCR

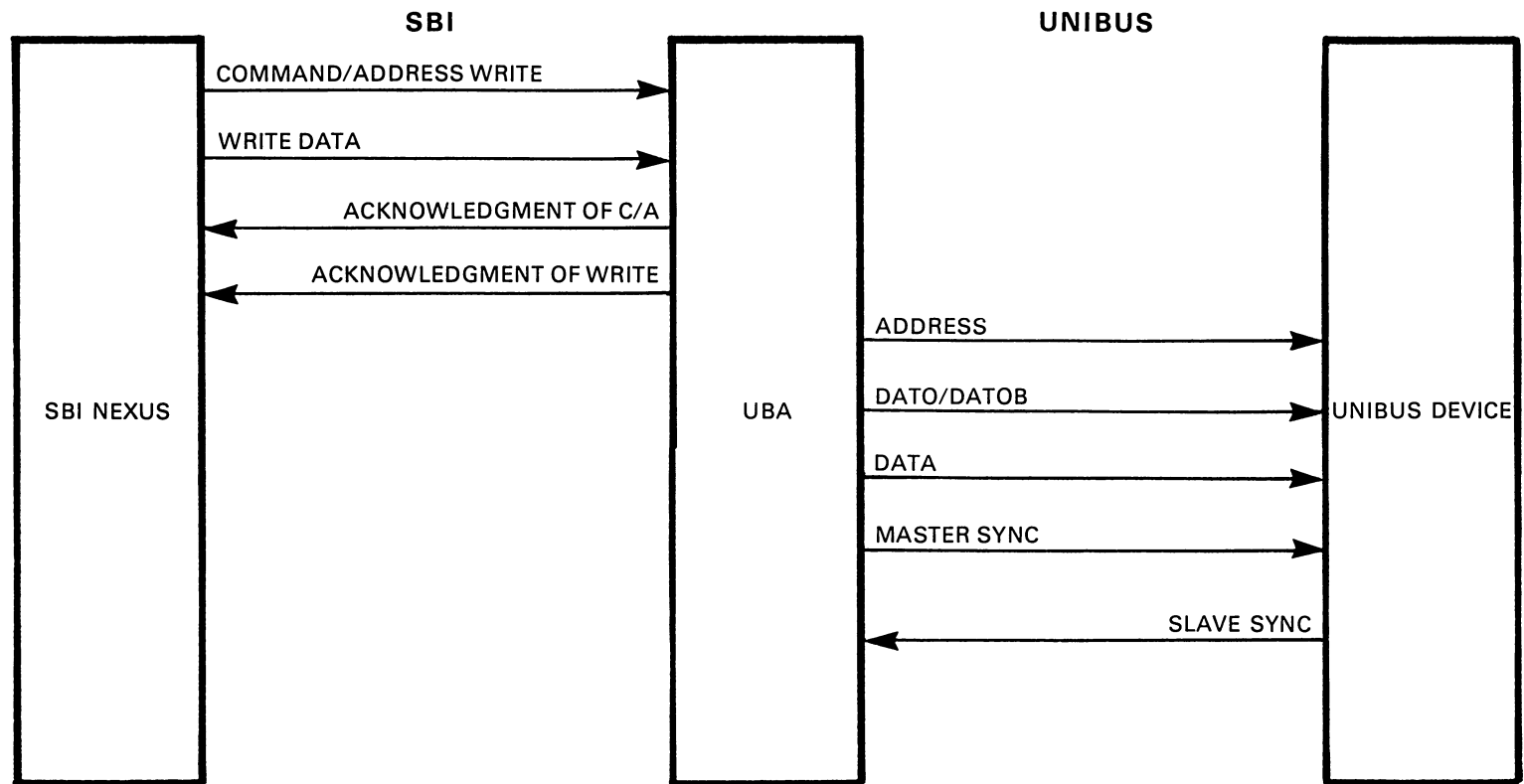
TK-3610

READ DCR
DOES NOT ENVOKE
MICRO SEQUENCER



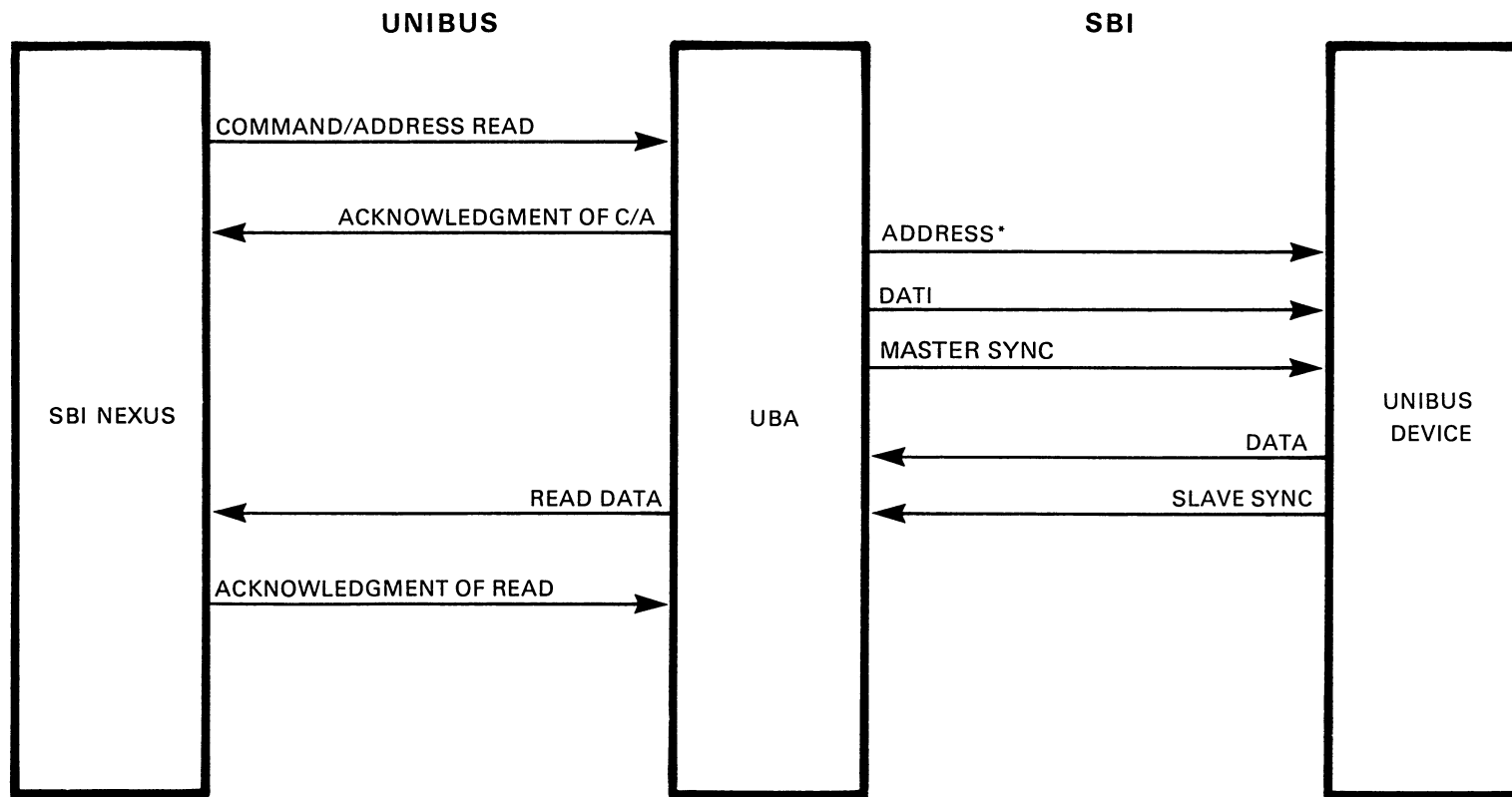
Read DCR

TK-3609



TK-3584

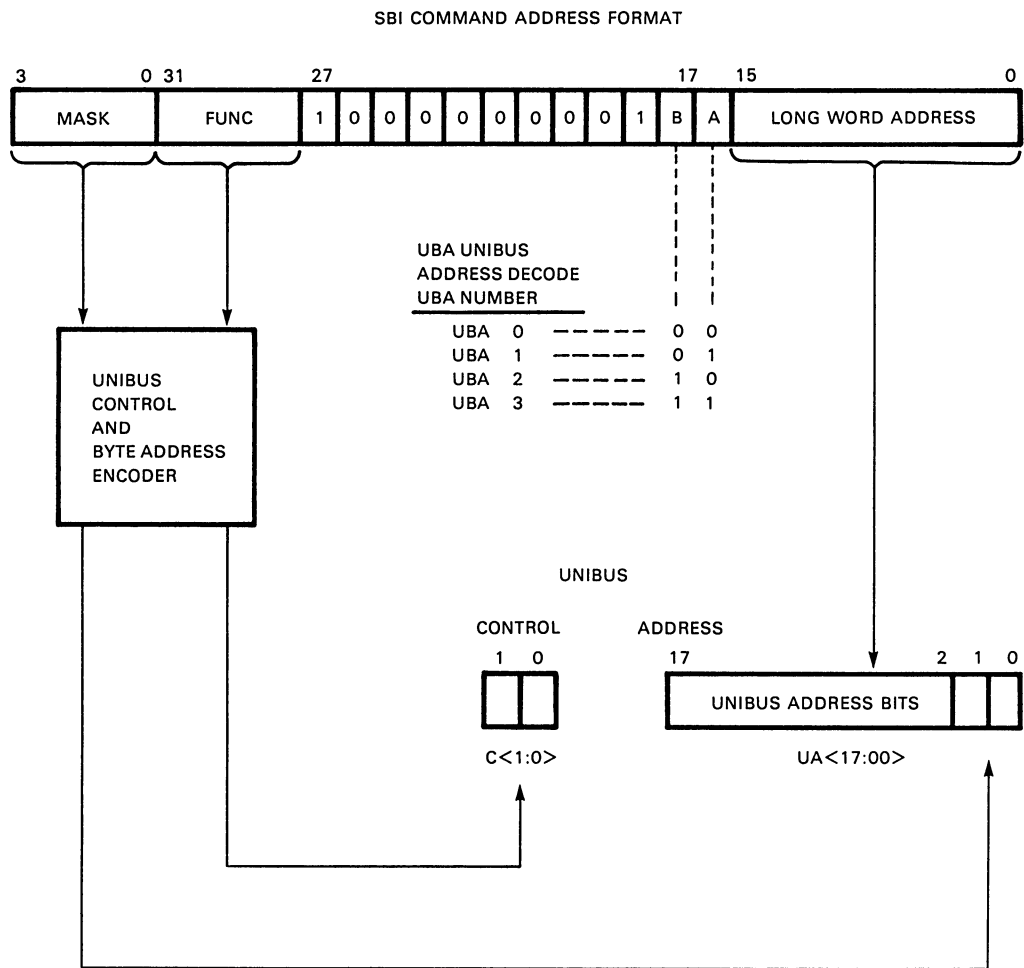
SBI to UNIBUS Device Write



*SEE SBI TO UNIBUS ADDRESS TRANSLATION DIAGRAM

TK-3586

SBI to UNIBUS Device Read

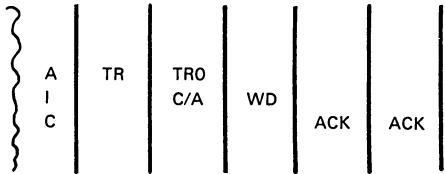


TK-3598

SBI to UNIBUS Control Address Translation

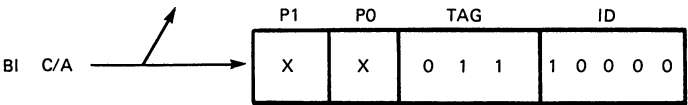
CPU WRITE TO
UNIBUS REGISTERS
(EK611 CSR #1)

PA = 2013FF20
SBI=804FFC8
UBA=777440₈



	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PA	0	0	1	0	0	0	0	0	0	0	0	1	0	0	1	1	1	1	1	1	1	1	1	1	0	0	1	0	0	0	0	0

MASK				FUNC				27																									
0	0	1	1	0	0	1	0	1	0	0	0	0	0	0	0	0	1	0	0	1	1	1	1	1	1	1	1	0	0	1	0	0	0



B3				B2				B1				B0			
← IGNORED →								← SHOULD CONTAIN THE INFORMATION TO BE WRITTEN INTO CSR#1							
P1		P0		TAG		ID				MASK					
X		X		1 0 1		1 0 0 0 0				0 0 1 1					

TK-3611

TK-3611

NOTE

DEVICE REG ADDRESS
ALLOCATION

UNIBUS ADDRESSES 760 000 to 777777₈
PA 201 X E000₁₆ TO 201 X FFFF₁₆
X = 3, 7, B, or F DEPENDING ON UBA IN QUESTION

MICRO SEQUENCER
NOT
ENVOCKED

DATA IS TRANSFERED THRU THE "I" BUS TO
CLK IN IB LINE RCURS. THRU MUX THAT
SELECTS LOW WORD & LOADS DATA INTO WD
STORE & WHEN THE UNIBUS IS AVAILABLE
SHIFT THE DATA OUT.

ADDRESS IS HELD IN A & C HOLD (UAIH, P)

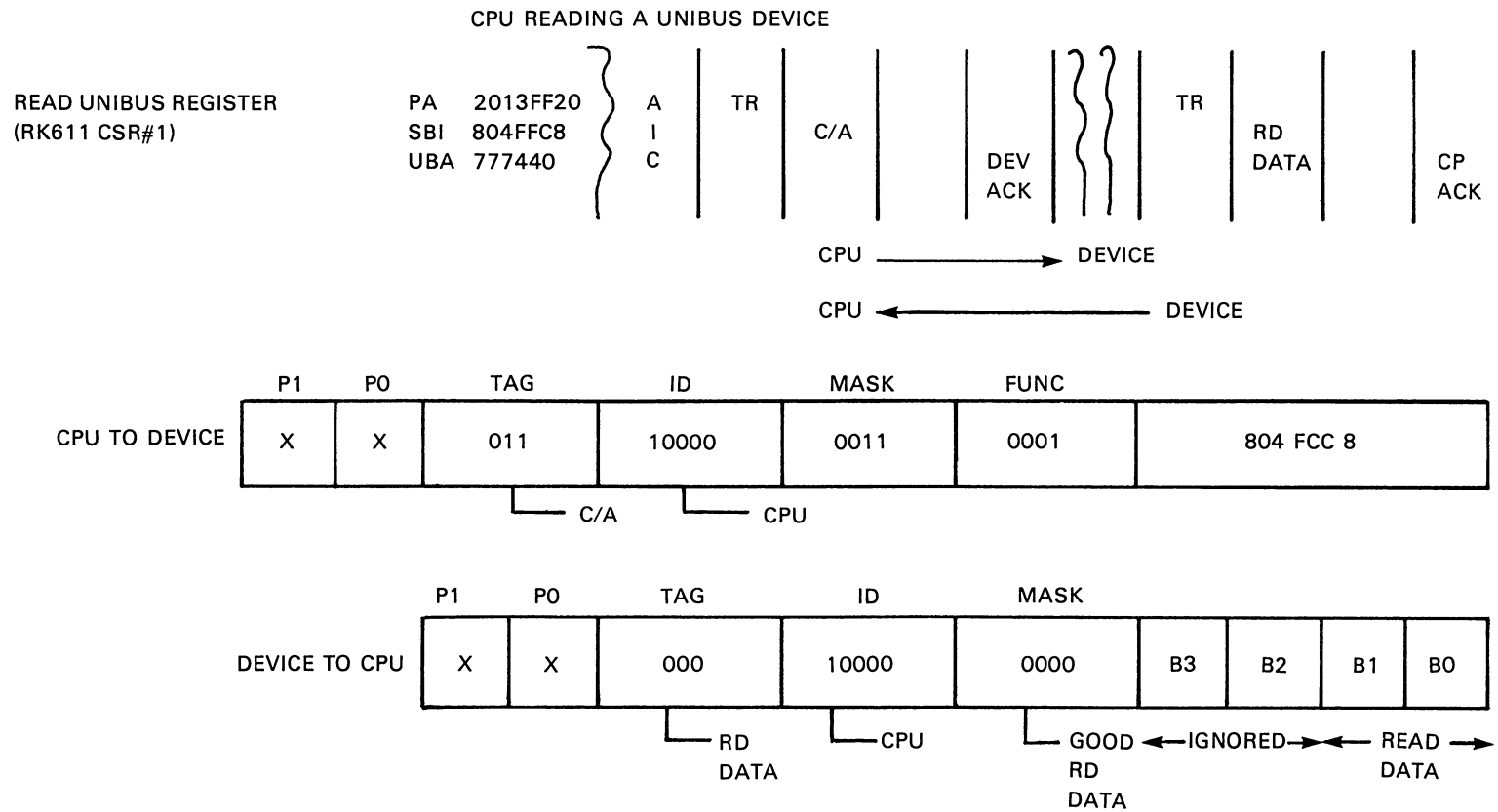
CPU Write to UNIBUS Register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	ROM WORD BIT NUMBER
BRANCH OFFSET						NEXT ADDRESS						FIELD				
	F	E	D	C	B	A	8	7	6	5	4	3	2	1	0	FIELD BIT NUMBER
SPR	BRG			BRS			NAD						ROM FIELD MNEMONICS			

43424140393837363534333231302928272625242322212019181716																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																													
CONT FIELD D																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																													

TK-0100

Microsequencer ROM Fields



MICRO SEQUENCER
IS
ENVOKED
UB ATT

DATA FROM THE UNIBUS IS BROUGHT
THRU THE SHIFTER, BUT NOT THRU THE
BUFFER DATA PATH RAMS, INSTEAD IT IS
FED THRU THE IB OUTMUX ONTO THE IB
BUS.

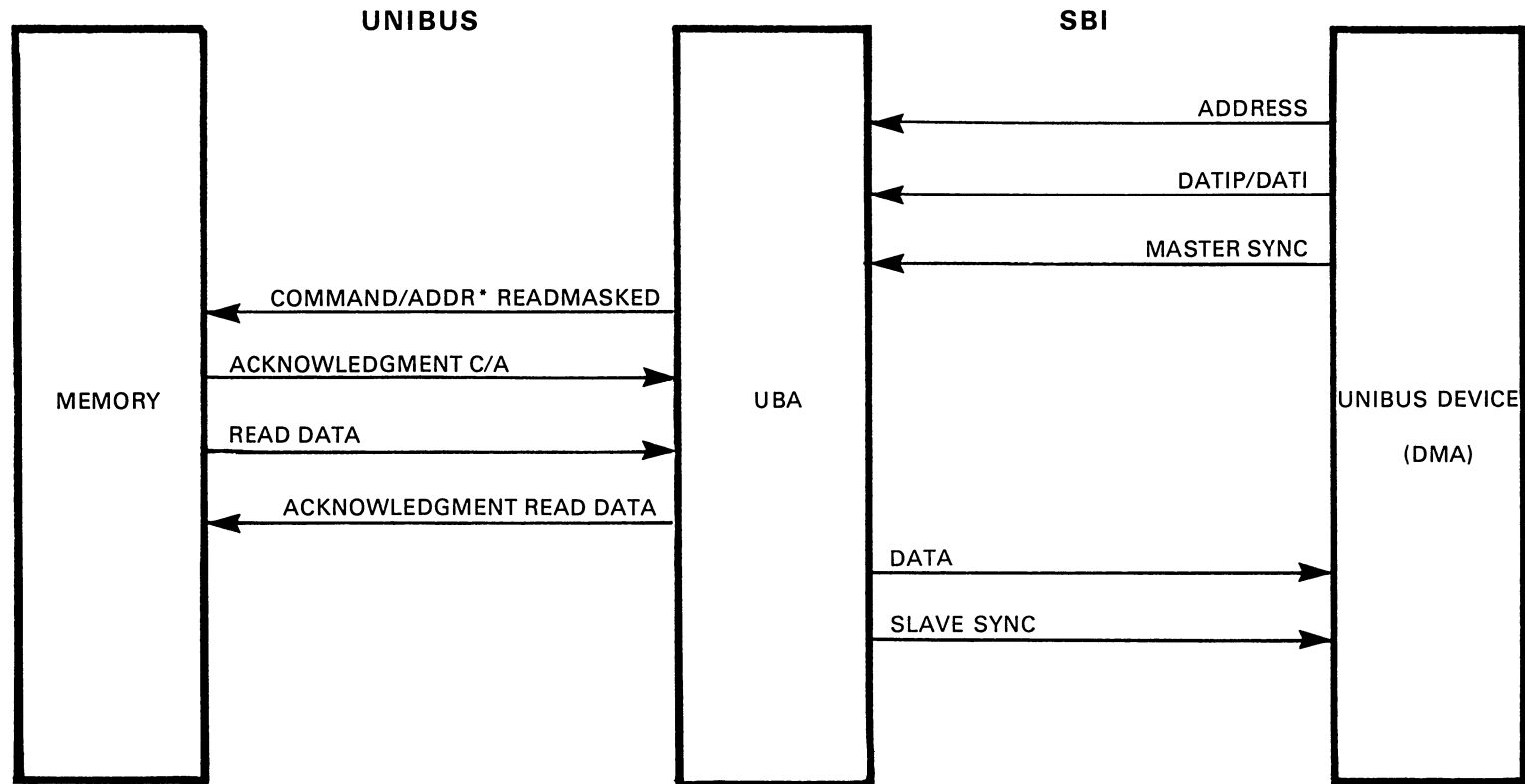
USE FLOWS
μADR 52, 53
&
BLOCK DIAGRAM

CPU GENERATES C/A

UBA ADAPTER GENERATES AN NPR TO
GET BUS

TK-3597

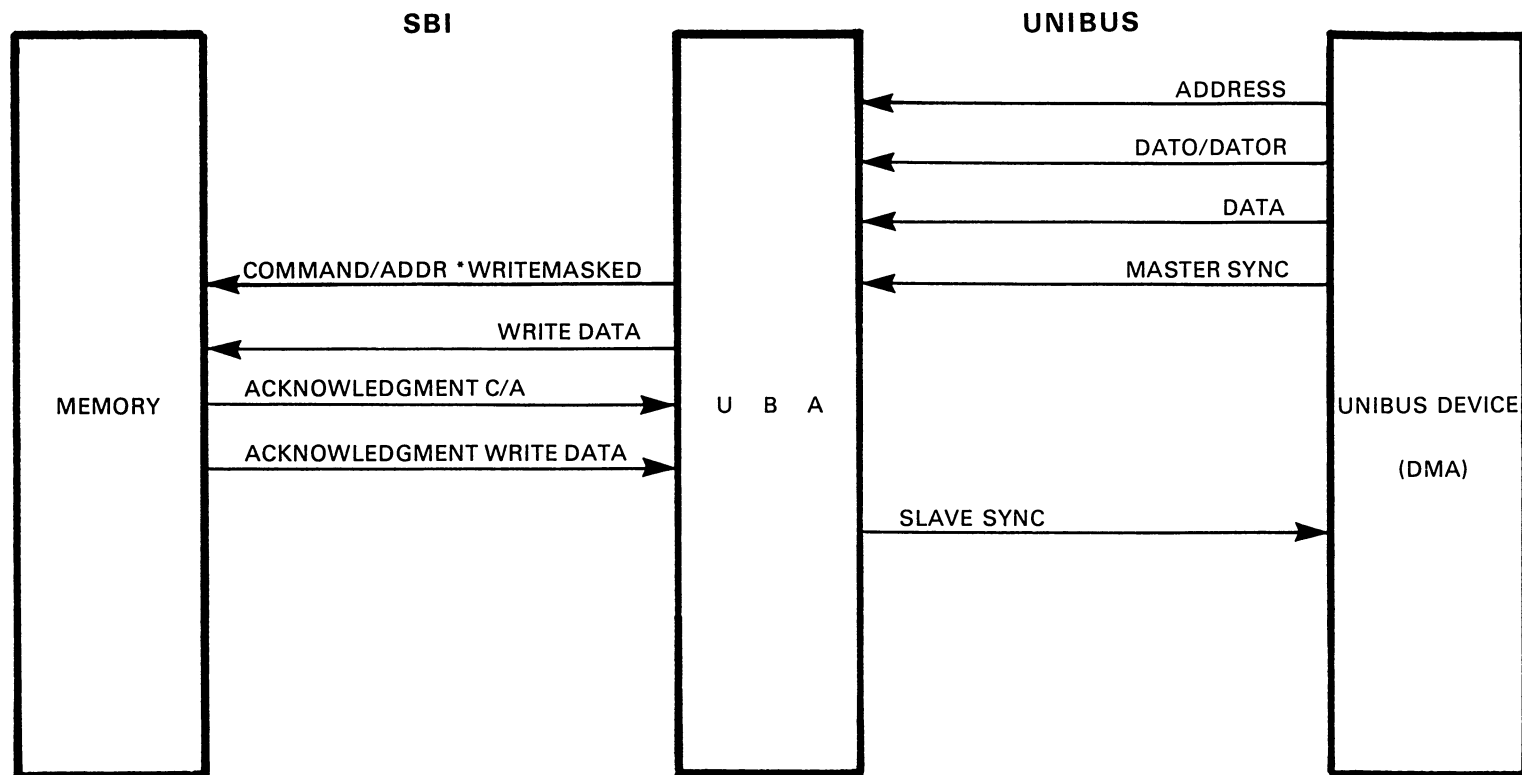
Read UNIBUS Register



* SEE UNIBUS TO SBI ADDRESS TRANSLATION (DP# = DPO)

TK-3587

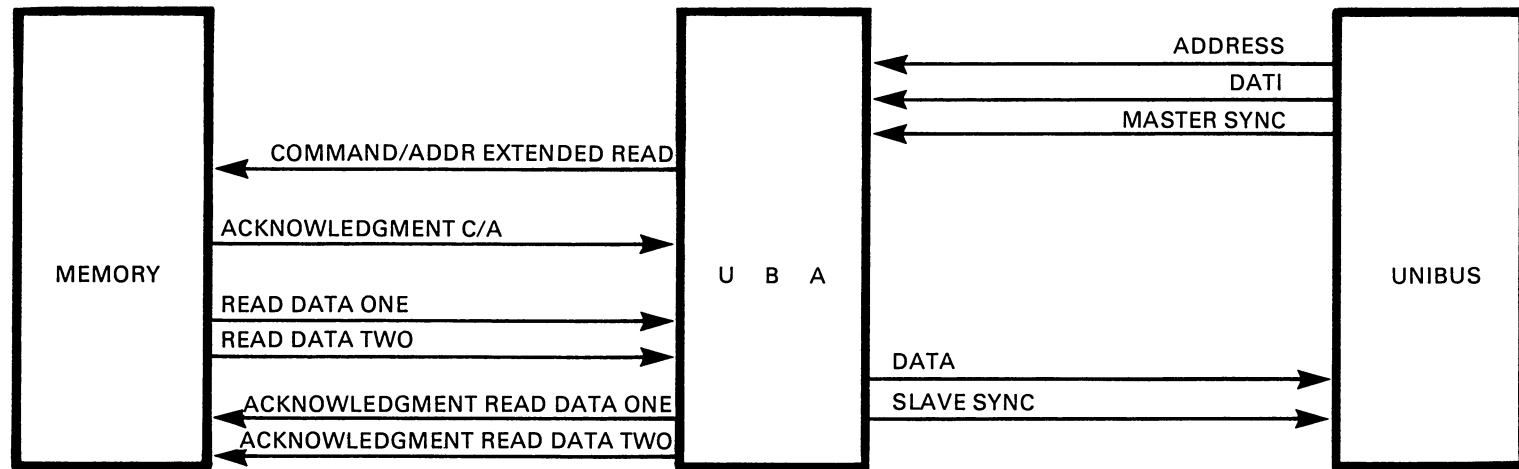
UNIBUS to SBI Through Direct Data Path



* SEE UNIBUS ADDRESS TO SBI ADDRESS TRANSLATION DIAGRAM (DP#=DPO)

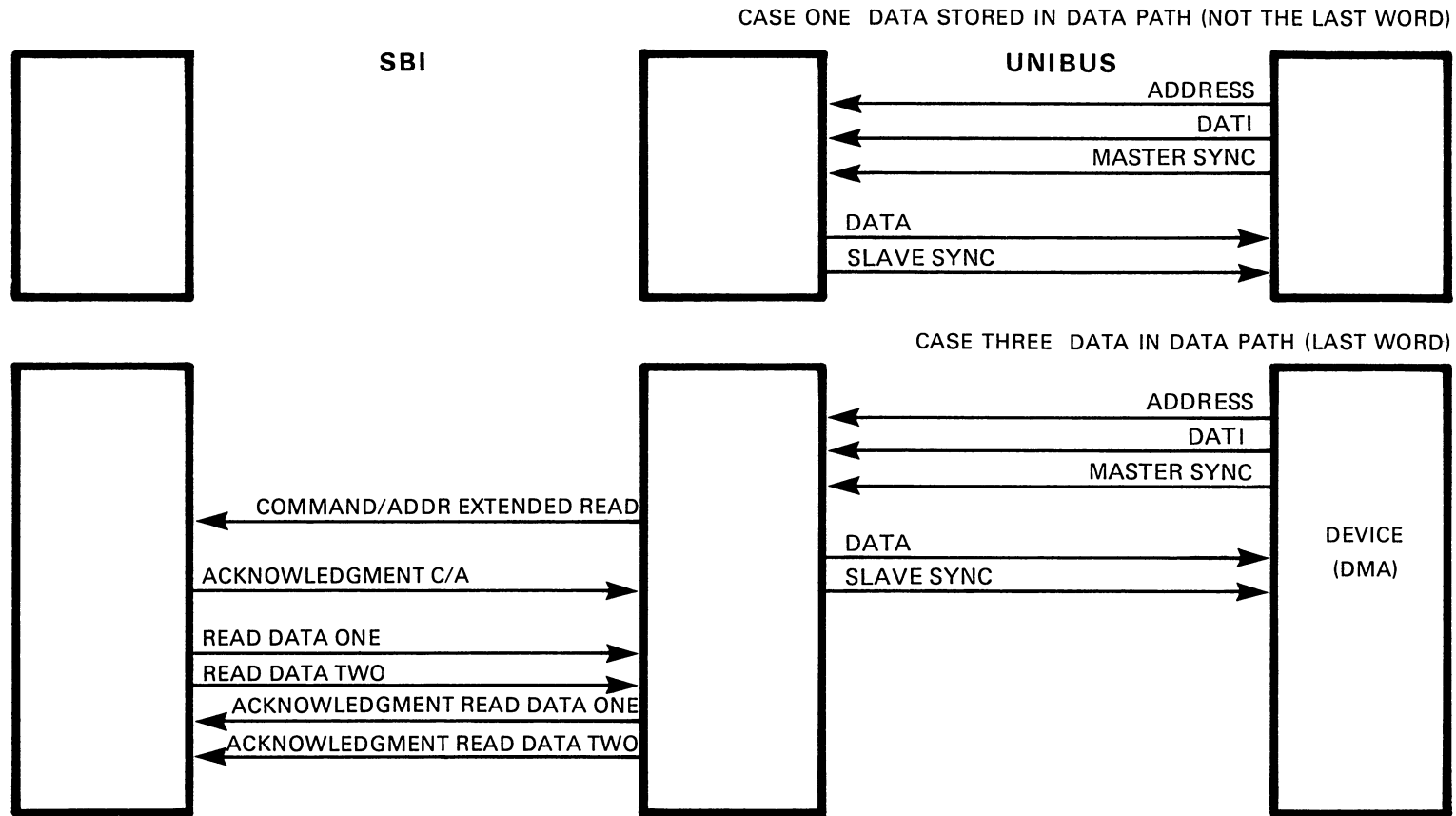
TK-3588

UNIBUS to SBI DMA Write Through Direct Data Path



TK-3589

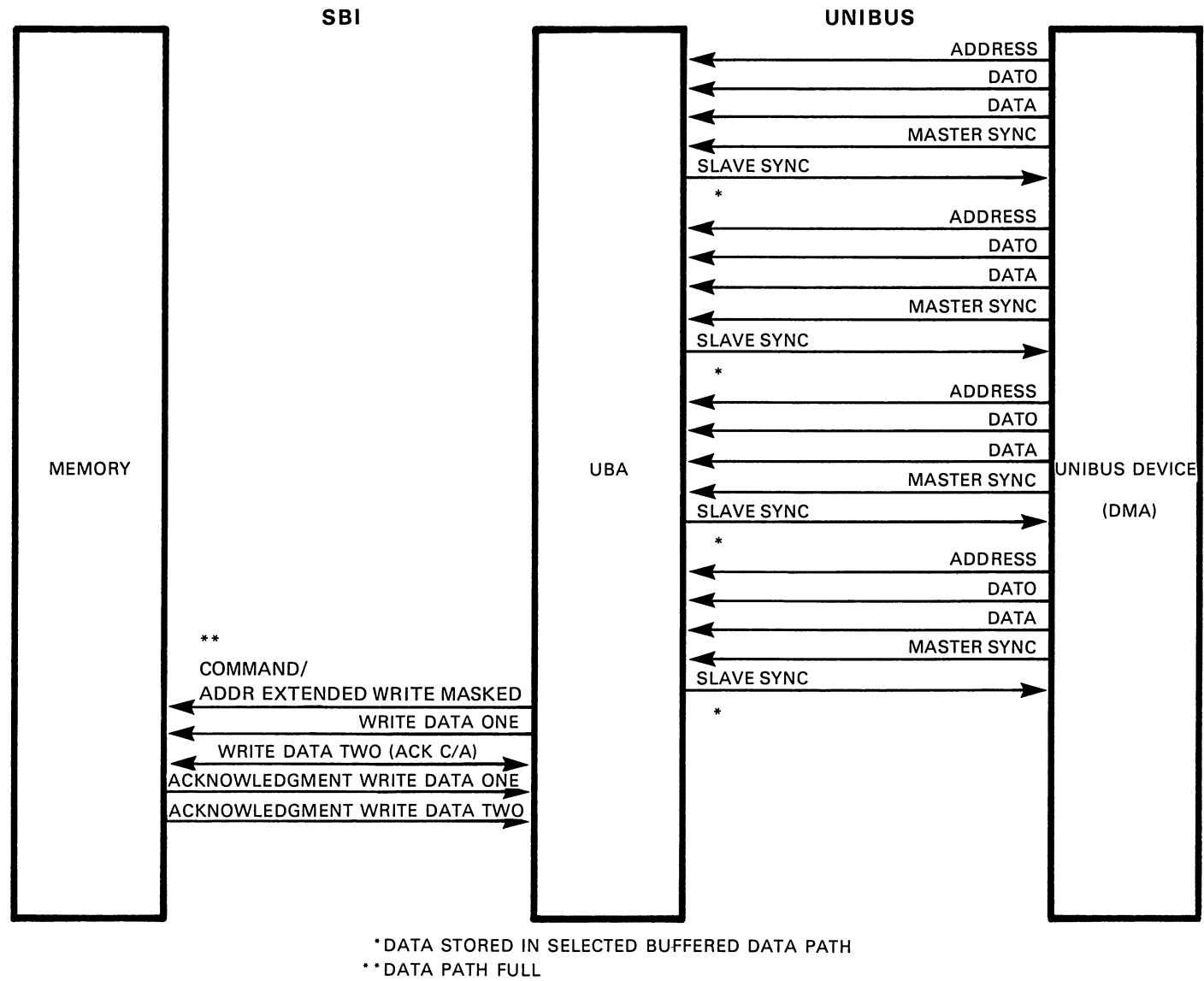
UNIBUS to SBI Read Through Buffered Data Paths



*ALL FOUR WORDS STORED IN DATA PATH
 **PRE-FETCH OF NEXT FOUR WORD

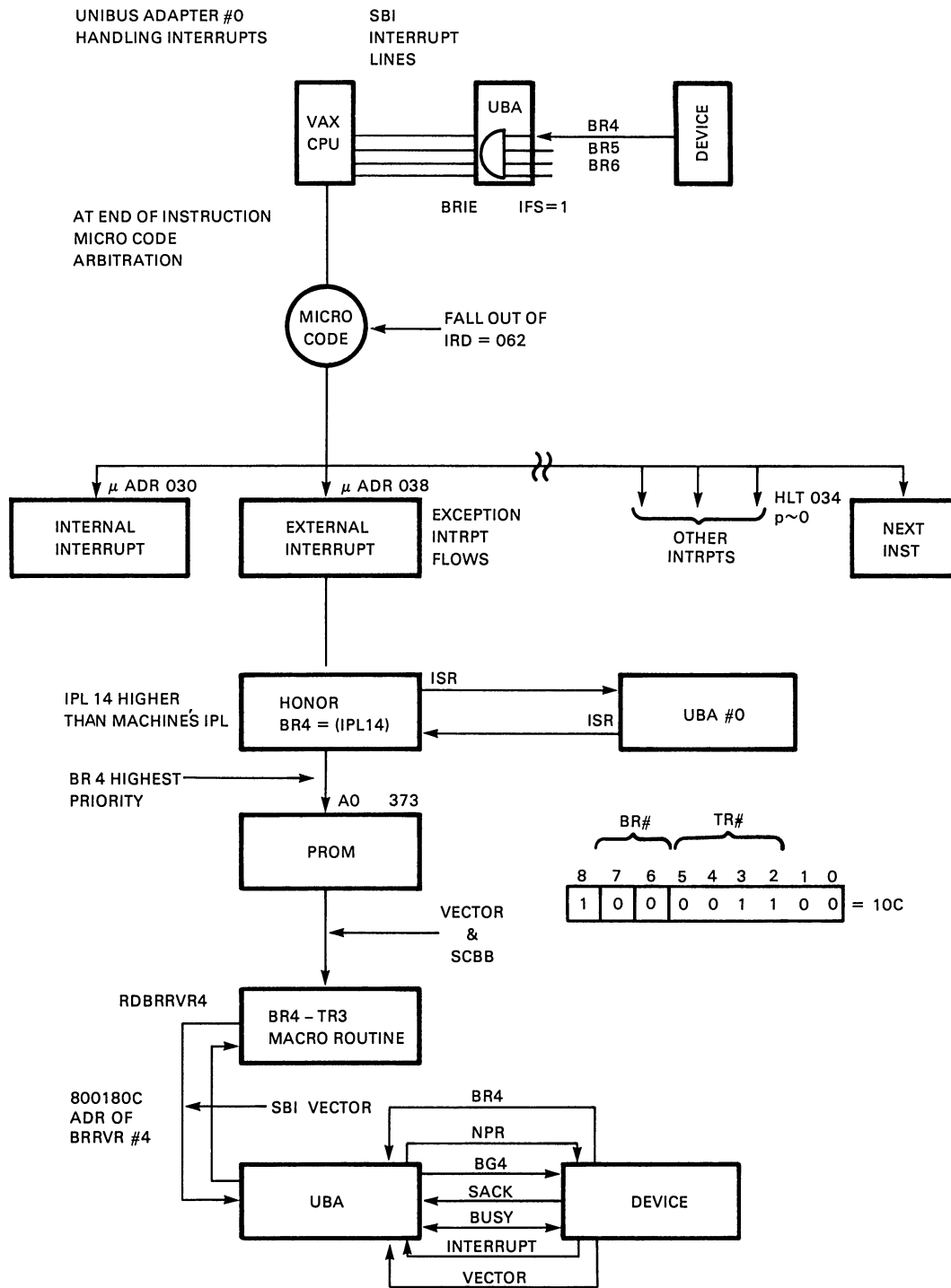
DMA Transfers

TK-3591

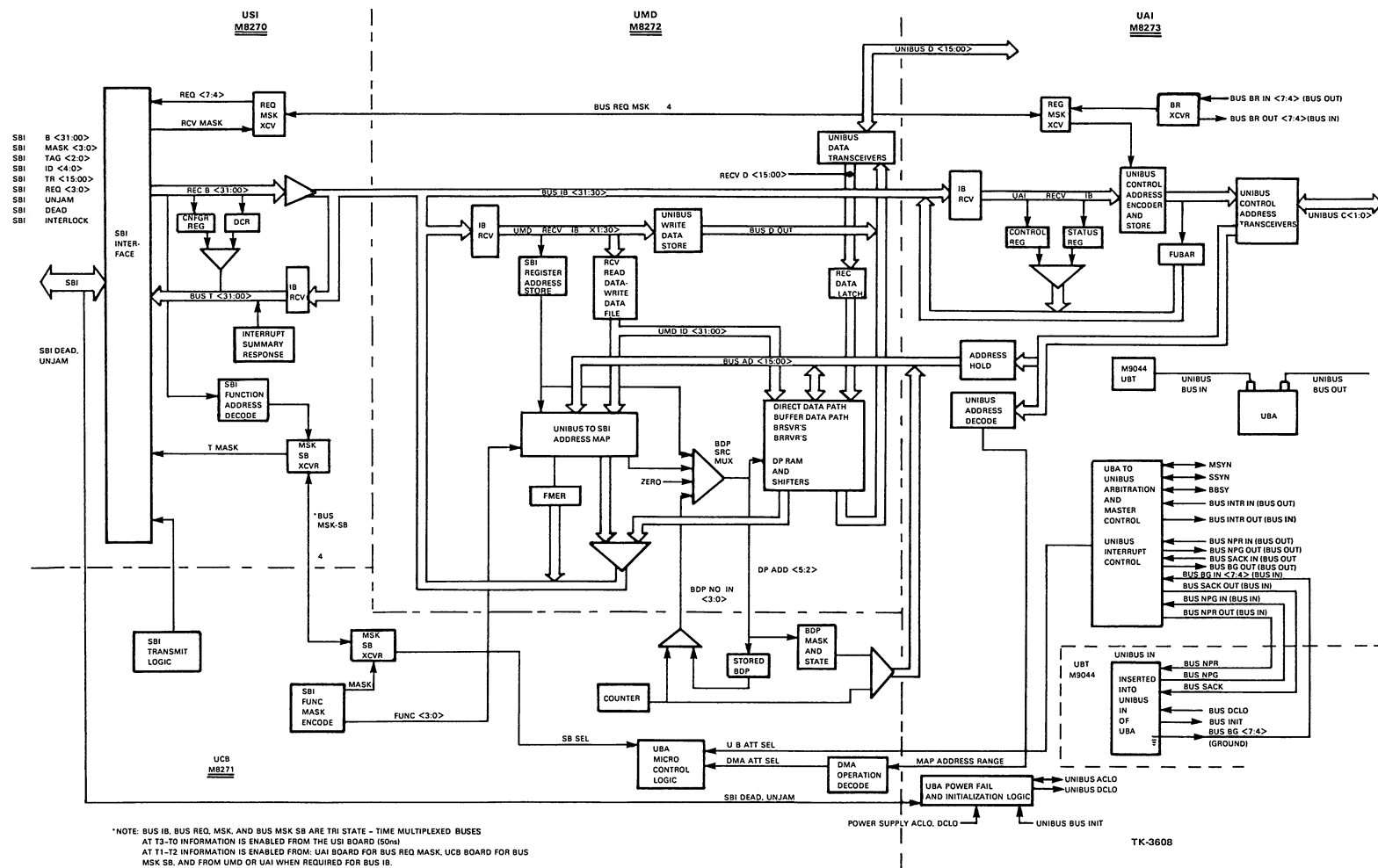


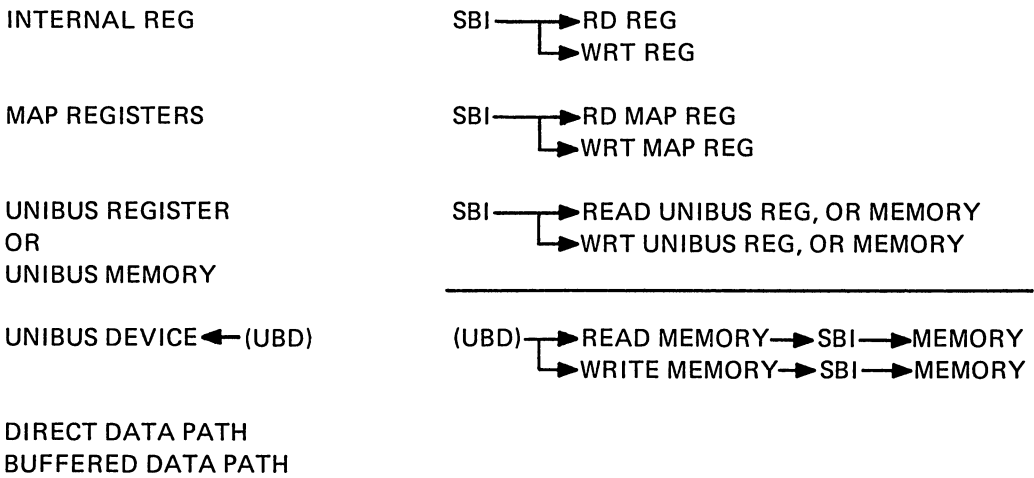
UNIBUS DMA Write Through Buffered Data Paths

TK-3592



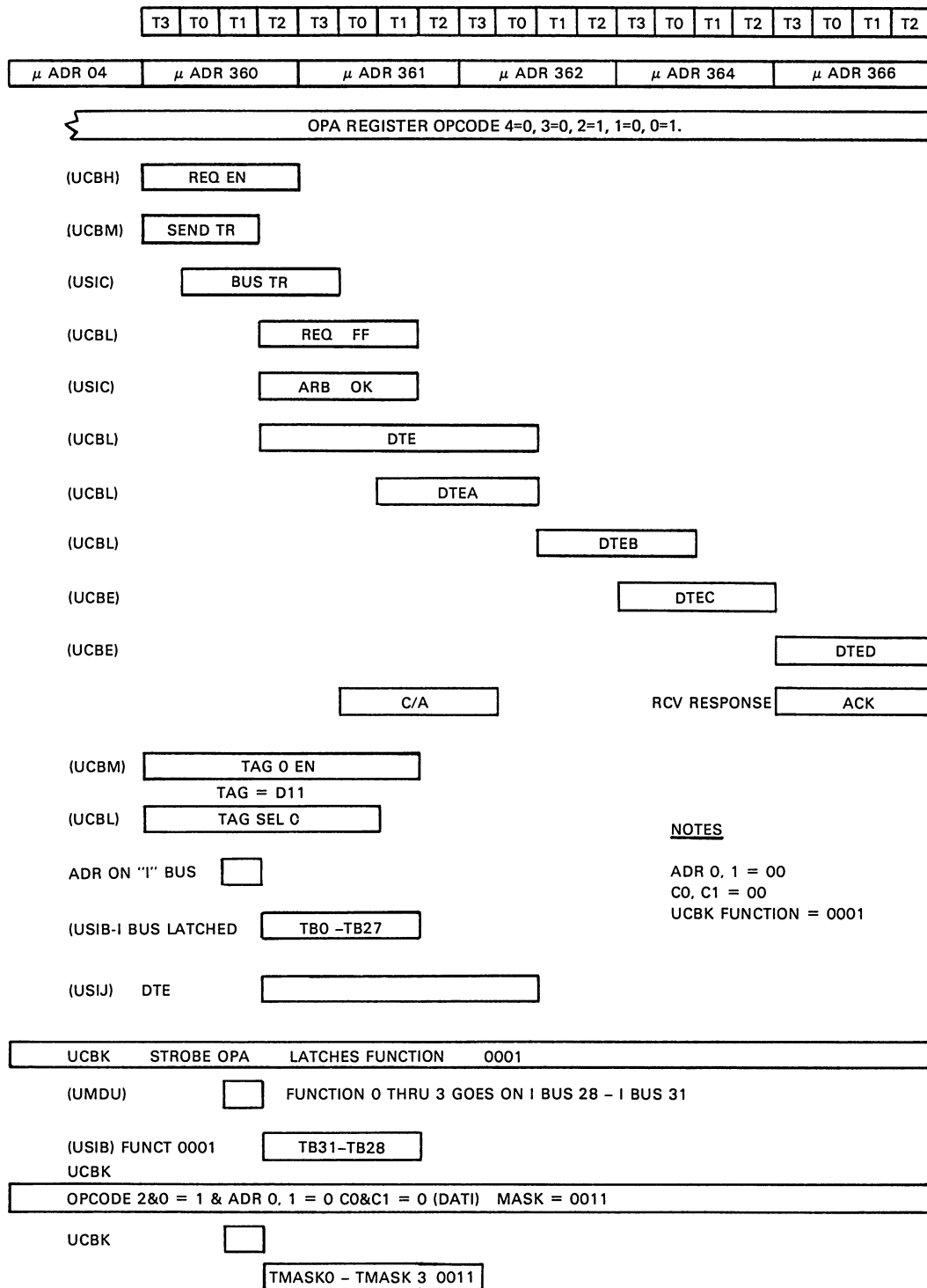
UBA Interrupts





TK-3853

Types of UBA Operations



NOTES

ADR 0, 1 = 00

C0, C1 = 00

UCBK FUNCTION = 0001

TK-3594

Microcode Example for DDP Read

MODULE TEST

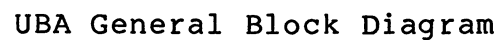
1. Select from the list below all those functions that require the aid of the microsequencer by placing an X to the left of the function:

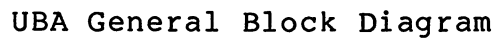
____ Read configuration register
____ Read any BRRVR register
____ Write any map register
____ Write status register
____ Read the control register
____ Write configuration register
____ Read any map register
____ Read status register
____ Write control register
2. Given below is one function which requires the use of the microsequencer. List the microsequence required to execute it. Assume no errors will occur.

a. Data path register 5 read
3. On the copies of the UNIBUS functional block diagrams provided, draw the data path needed to perform the following data transfers between the SBI and UNIBUS.

a. VAX-11/780 reads the DW780 map register 1.

b. VAX-11/780 reads a register of a UNIBUS device.





BLANK

BLANK



EDUCATIONAL SERVICES